Entwicklung einer Steuersoftware des IBIS-Experiments auf INTEGRAL

Diplomarbeit

vorgelegt von

Eckart Göhler

Eberhard–Karls–Universität Tübingen
Institut für Astronomie und Astrophysik
— Abt. Astronomie —

April 1999

Inhaltsverzeichnis

Einleitung

Per aspera ad astra
Seneca, Herc.f.437

Satelliten stellen der modernen Astronomie ein Instrumentarium zur Verfügung, das ihr Beobachtungen ermöglicht, die jenseits terrestrischer Messungen liegen. Denn Astronomie ist weitgehend auf die Detektion elektromagnetischer Strahlung angewiesen, von der nur ein Bruchteil ungehindert durch die Erdatmosphäre gelangt. An diesen nun nicht mehr ganz neuen Typ von Instrumenten werden aber gerade wegen seiner Erdferne besonders hohe Anforderungen gestellt, da eine Störung oder gar Versagen des Messbetriebes äußerst kostspielig ist.

Die vorliegende Diplomarbeit beschreibt die Entwicklung einer Software, die die Steuerung verschiedener Teile eines Gamma-Detektors namens **IBIS** übernimmt. Dieses Instrument befindet sich mit anderen Instrumenten auf dem **INTEGRAL**-Satelliten, der Spektrum, Richtung und Zeitverhalten der Gammastrahlung im Bereich von etwa 15 keV bis 10 MeV im Kosmos untersuchen soll.

Am Astronomischen Institut in Tübingen (IAAT) sind bereits einige Arbeiten für diesen Satelliten entstanden, die sich vorzugsweise mit einer Datenverarbeitungselektronik (HEPI) beschäftigt haben, die den rechenintensiven Teil der An–Bord–Auswertung übernimmt. Die Software, die in einem Computer die weitere Datenverarbeitung an Bord von Integral vornimmt, ist zusammen mit Martin Stuhlinger entwickelt worden.

Die Arbeit gliedert sich im wesentlichen in drei Teile. Im ersten Teil soll das **IBIS**-Instrument mit allen Komponenten, seinen wissenschaftlichen Möglichkeiten, der Hardware und den Software-Anforderungen vorgestellt werden (Kapitel 1 bis 3).

Der zweite Teil beinhaltet die Struktur der Software, d.h. ihren Aufbau, ihre Funktionen und die Algorithmen (Kapitel 4 bis 6).

Der dritte Teil setzt sich mit dem Software-Entwicklungsprozess auseinander. Hier sollen die Schwierigkeiten und Lösungen, die sich bei der Umsetzung der Anforderungen in ein konkretes ADA-Programm ergaben, beschrieben werden. Dazu gehört auch der schon in den vorigen Kapiteln angerissene Validierungsprozess der Software, der verschiedene Aspekte der Fehleranalyse zusammenfasst (Kapitel 7).

Kapitel 1

Der INTEGRAL-Satellit

1.1 Die Mission des INTEGRAL–Satelliten

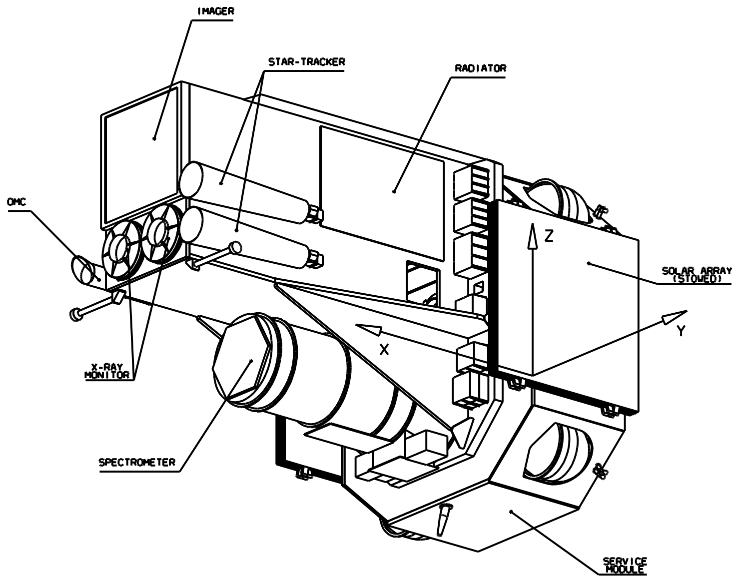

Abbildung 1.1: Der INTEGRAL–Satellit

In der Gamma–Astronomie lieferten bereits viele Instrumente in Ballonen und Satelliten manigfaltige Ergebnisse. In neuerer Zeit sind dies Experimente wie COMTEL des Compton Gamma–Ray–Satelliten (CGRO) und SIGMA auf GRANAT, die im weichen Gammabereich weitere neue Erkenntnisse brachten. Die Auswertung der Ergebnisse zeigte jedoch, dass zur Klärung vieler Fragen ein Instrument mit einer höheren Orts– und Energieauflösung nötig ist, als es bisher zur Verfügung stand [1].

Aus diesem Grund wurde 1993 von dem wissenschaftlichen Komitee der ESA als nächstes mittelgroßes Projekt die INTEGRAL–Mission in Angriff genommen, die diesem Mangel abhelfen soll. Der **INTEGRAL–Satellit**, der im Frühjahr 2001 mit einer PROTON–Rakete (als Ausweichlösung ist eine ARIA-

NE V vorgesehen) starten soll und für eine Betriebszeit von nominal zwei Jahren ausgelegt ist, soll im Energiebereich von 3 keV bis 10 MeV arbeiten und sowohl eine Himmelskartierung wie auch hochaufgelöste Einzelbeobachtungen zulassen. Um lange Beobachtungszeiten zu erreichen wird der INTEGRAL-Satellit auf eine stark exzentrische Umlaufbahn mit maximal (je nach Trägerrakete) 10000 km Perigäum, 153000 km Apogäum[1] und einer Umlaufzeit von etwa 72 Stunden gebracht. Je nach Umlaufbahn kann der Satellit dabei von der Erde abgeschattet werden (Eklipse) oder in den Strahlungsgürtel der Erde eintauchen. Mehrere Bodenstationen sichern einen ständigen Kontakt zum Satelliten, der seine Messergebnisse ohne größere Zwischenspeicherung zur Erde sendet [7].

1.2 Wissenschaftliche Möglichkeiten von INTEGRAL

1.2.1 Die Instrumente des INTEGRAL–Satelliten

Der INTEGRAL–Satellit trägt vier Instrumente:

- Zwei abbildende Teleskope für die Bereiche 20 keV–8 MeV (SPI) und 15 keV–10 MeV (IBIS). SPI ist als Spektrometer mit hoher Energieauflösung ausgelegt. IBIS dagegen ist als abbildendes Teleskop mit hoher Ortsauflösung vorgesehen.
- Einem abbildenden Röntgenmonitor für den Bereich 3 keV–60 keV (JEM-X).
- Eine optische Kamera (OMC).

In Tabelle 1.1 sind die wesentlichen Eigenschaften dieser Instrumente dargestellt. IBIS und SPI sollen sich in ihren Eigenschaften (hohe Energie- vs. Ortsauflösung) ergänzen und decken etwa den gleichen Energiebereich im weichen Gammaspektrum ab. JEM–X soll dagegen die Messung im Röntgenbereich, in dem höhere Intensitäten zu erwarten sind, überwachen. OMC soll helfen, die sichtbare Komponente eines Gamma-Objektes zu finden. Alle Instrumente sind daher stets auf dasselbe Objekt gerichtet, so dass sich ihre komplementären Eigenschaften ergänzen können. Jedes Instrument hat seine eigene Datenverarbeitung, die die Daten dem Hauptrechner von INTEGRAL, der CDMU (Central Data Management Unit), übergibt.

1.2.2 Anwendungsmöglichkeiten

INTEGRAL ist für die Untersuchung nuklearer Zerfallsprozesse gut geeignet, da IBIS und SPI gerade in diesem Energiebereich sensitiv sind. Es bietet sich daher eine Kartierung der galaktischen Ebene im Gammabereich an, um Bereiche mit verschiedenen Sternentstehungsprozessen festhalten zu können.

Genauso bedeutend ist die Beobachtung anderer stellarer Strahlungsprozesse, wie etwa der Annihilationsstrahlung der Positroniums, der Untersuchung von Zyklotronstrahlung und der Temperaturstrahlung kompakter Objekte. Wegen

[1] [7, Seite 2.2].

	SPI	IBIS	JEM–X	OMC
Energiebereich	20 keV- 8 MeV	15 keV- 10 MeV	3 keV- 100 keV	500- 850 nm
Detektorfläche (cm^2)	500	2600 (CdTe) 3100 (CsI)	1000 (2×500)	2048×1024 Pixel
Spektrale Auflösung $\Delta E/E$	0.2% (1 MeV)	6% (1 MeV)	5% (E> 35 keV)	—
Gesichtsfeld	16°	$9° \times 9°$	4.8°	$15° \times 15°$
Winkelauflösung	2°	$12'$	$3'$	$17.6''$/pixel
Zeitauflösung	0.1 msec	< 1 msec	10 μsec	~ 1 sec

Tabelle 1.1: Schlüsselparameter der Instrumente auf INTEGRAL [7, Seite 2.3]

der guten Energie– und Ortsauflösung ist INTEGRAL besonders für Untersuchungen innerhalb der galaktischen Ebene geeignet, wobei IBIS mit seinem hohen Auflösungsvermögen bei der Lokalisierung und Identifizierung von stellaren Objekten helfen kann. Auch extragalaktische Objekte sind, sofern sie im Gammabereich eine ausreichende Intensität haben, lohnende Ziele für INTEGRAL–Experimente. In [1] und [8, 17] sind eine Vielzahl von Anwendungsmöglichkeiten beschrieben.

1.3 Aufbau des INTEGRAL–Satelliten

1.3.1 Service Module

Die Instrumente bilden das sogenannte **Payload Module**. Sie sind auf eine Plattform, dem **Service Module**, montiert, welches auch für den Satelliten XMM Verwendung findet. Das Service Module steuert die Satellitenausrichtung, trägt die Solarpanele für die Energieversorgung und stellt beim Start die Verbindung zum Raketenmodul her [1].

1.3.2 Die kodierte Aperturmaske

Anders als im optischen Bereich gibt es in der Gammaastronomie keine fokussierende Abbildung, weil für Wellenlängen kleiner als Atome keine beugenden oder brechenden Materialien existieren. Auch die im weichen Röntgenlicht verwendete Methode der streifenden Reflektion, die erstmalig im Satelliten ROSAT Verwendung fand, kann nicht mehr benutzt werden.

Für eine Richtungsbestimmung gibt es andere Möglichkeiten. Die einfachste besteht aus Kollimatoren, die alle Strahlen außerhalb der Beobachtungsrichtung ausblenden. Der Nachteil ist die geringe Empfindlichkeit, die aber durch geeignete Maßnahmen erhöht werden kann [9].

Eine weitere Möglichkeit ist die Benutzung der Compton–Streuung, bei der aus dem Energieverlust einer Streuung die Ursprungsrichtung rekonstruiert werden kann.

Bei INTEGRAL findet in den Instrumenten IBIS, SPI und JEM–X die Methode der **kodierten Maske** Anwendung. Dazu wird das Gesichtsfeld des

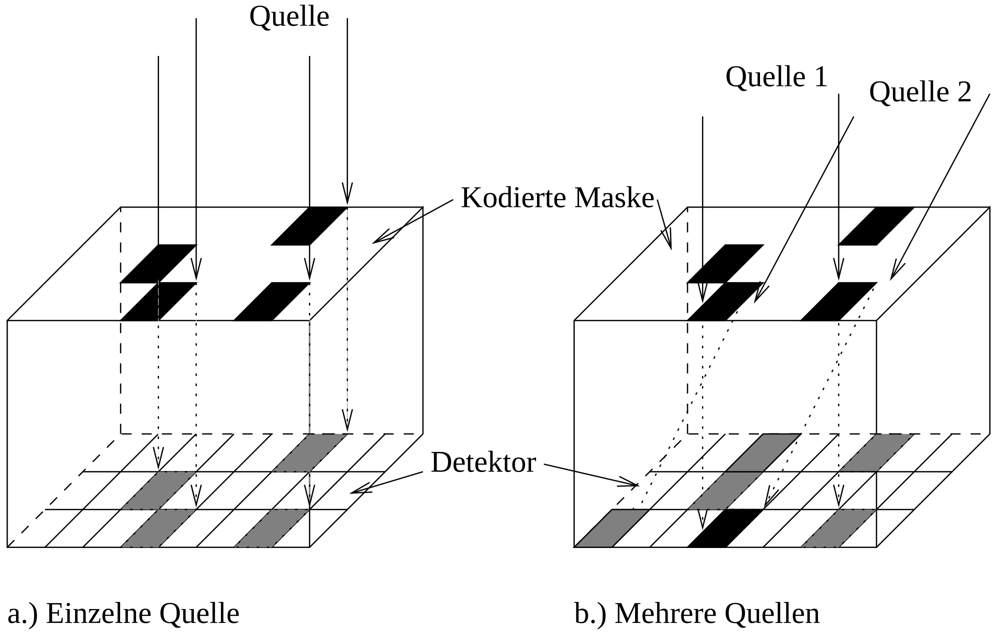

Abbildung 1.2: Schema der kodierten Maske

Detektors mit einer gemusterten Maske bedeckt. Die zu untersuchende Strahlungsquelle projiziert dieses Muster auf den Detektor, der aus einem Feld von Einzelelementen (Pixel) besteht. Aus dem beobachteten Schattenwurf kann die Richtung der Strahlungsquelle bestimmt werden. Bei mehreren Objekten überlagern sich die Projektionen. Dennoch ist eine Rekonstruktion möglich, sofern das Maskenmuster geeignet gewählt wird [1]. Schwierigkeiten ergeben sich allerdings, wenn nicht die ganze Maske auf das Detektorfeld abgebildet wird. In diesem Fall befindet sich die Quelle nicht mehr im **vollkodierten Blickfeld** (Full Coded Field of View — FCFOV).

1.4 Das Spektrometer auf INTEGRAL (SPI)

Das Spektrometer **SPI** (Spectrometer on Integral) besteht aus 19 hexagonalen Germaniumdetektoren, die auf etwa 85 K gekühlt werden müssen. Für eine Ortsbestimmung sind die Detektoren außerhalb des Gesichtfeldes mit einer auf antikoinzidenz geschaltenen Detektorummantelung aktiv abgeschirmt.

Die Energieauflösung ist mit etwa 1.38 keV bei 511 keV sehr hoch und erlaubt, dass im Gammabereich aus dem Linienprofil die Gastemperatur oder aus der Linienverschiebung die Entfernung einer Quelle messbar ist. Die Sensitivität von SPI ist im wesentlichen durch die Hintergrundstrahlung begrenzt; sie hängt demnach von der Effektivität der Abschirmung ab.

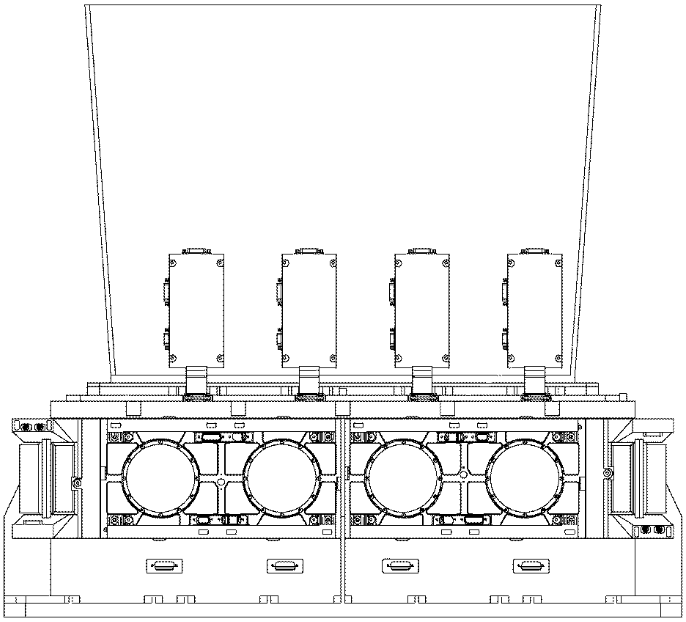

Abbildung 1.3: Die Detektoren von IBIS mit Abschirmung

1.5 Der Imager auf INTEGRAL (IBIS)

Das **IBIS–Instrument** (Imager On Board Integral Satellite) verwendet zwei übereinanderliegende Detektoren namens ISGRI und PICsIT, die in unterschiedlichen Energiebereichen arbeiten.

1.5.1 Die Aperturmaske von IBIS

Die Ortsinformation wird auch hier mit einer kodierten Aperturmaske, die aus 95×95 Einzelzellen mit je 11.2×11.2 mm^2 besteht, gewonnen. Sie ist aus Wolfram gefertigt, so dass eine effektive Abschattung von 60% bei 1.5 MeV möglich ist. Die Transparenz der gesamten Maske ist im Energiebereich von IBIS etwa 50% [11]. Die Maske selber ist größer als die Detektorflächen, so dass sich ein vollkodiertes Gesichtsfeld von $9^\circ \times 9^\circ$ ergibt.

1.5.2 Abschirmung

Die Aperturmaske ist rund 3.1 m von dem oberen Detektor (ISGRI) entfernt. Um das Gesichtsfeld auf die Aperturmaske einzuschränken bzw. die Hintergrundstrahlung zu minimieren sind zwei Abschirmmethoden vorgesehen:

- Eine passive Abschirmung, die die Seitenwände von IBIS umgibt. Sie besteht aus einem trichterförmigen Wolframblech von 1 mm Dicke und einer röhrenförmigen Abschirmung aus Blei [11]. Diese Abschirmung ist allerdings nur bis zu Energien von etwa 200 keV wirksam.

- Eine aktive Abschirmung, die laterale und von unten kommende Strahlung ausblenden soll.

Die aktive Abschirmung, auch **VETO** genannt, besteht aus BGO–Blöcken (Wismut–Germanium–Oxid), an denen je zwei Photomultiplier befestigt sind. Wenn ein Photonenereignis oder Teilchenereignis in einem der Photomultiplier beobachtet wird, dann werden Ereignisse, die in den Detektoren gleichzeitig stattfinden, verworfen. Die aktive Abschirmung hat eine untere Schwelle von 100 keV, um die Totzeit der Abschirmung, die gerade wegen der hohen Rate niederenergetischer Photonen anwachsen würde, zu verringern.

1.5.3 CdTe-Detektor: ISGRI

Der obere Detektor, der auch ISGRI (Imager Soft Gamma–Ray on INTEGRAL) genannt wird, besteht aus 128×128 Pixeln. Jedes Pixel ist ein Kadmium–Tellur–Halbleiterdetektor. Diese Detektoren haben den Vorteil, dass sie bei 0–10° Celsius ihre optimale Temperatur haben und daher nicht gekühlt werden müssen. Auch gewährleistet die große Ordnungszahl einen hohen photoelektrischen Wirkungsquerschnitt im Gamma–Bereich. Wegen der relativ geringen Dicke von 2 mm nimmt die Empfindlichkeit für hohe Energien jedoch stark ab, so dass ISGRI den Energiebereich bis etwa 150 keV abdecken kann. Nachteilhaft ist bei diesem Typ von Detektoren, dass bei mittleren Energien Ladungen verlorengehen können (»trapping«), so dass die gemessene Energie zu niedrig ist. Aus der Anstiegszeit des Messsignals kann die tatsächliche Energie rekonstruiert werden. Die Korrektur ist allerdings nicht durch einen einfachen analytischen Ausdruck fassbar, was die Datenverarbeitung erschwert [27, 15]. In Tabelle 1.2 sind die wesentlichen Parameter von ISGRI festgehalten.

Elektronik der Module Die Elektronik von ISGRI ist auf acht einzelne Module (**MCE** – Module Control Electronics) verteilt, die weitgehend unabhängig voneinander arbeiten. Diese Elektronik kann unter anderem defekte Pixel erkennen und ausschalten, und sie verwaltet die Betriebsparameter von ISGRI. Siehe auch [15].

Pixelgröße:	4×4 mm^2	2 mm Dicke
Pixelabstand:	600 μm	(4.6 mm Abstand zweier Pixelzentren)
Effektive Fläche:	2621 cm^2	($128 \times 128 \times 16$ mm^2)
Arbeitstemperatur	0–10°C	
Energiebereich	15 keV–150 keV	
Abstand zur Aperturmaske:	3100 mm	
Zahl der Module:	8	

Tabelle 1.2: Schlüsselparameter von ISGRI

1.5.4 CsI-Detektor: PICsIT

Für höhere Energien ist ein weiterer Detektor (PICsIT— Pixellated Imaging CsI Telescope) vorgesehen, dessen Pixel aus Cäsium–Jodid–Szintillatoren bestehen. An das Szintillationsmaterial ist jeweils eine Photodiode optisch angekoppelt, die eine Verarbeitung der Szintillationsblitze ermöglicht.

Die Pixeldichte ist mit 64×64 Pixeln geringer als bei ISGRI. Die Energiebreite ist jedoch erheblich größer. Die Dicke der Pixel von 3 cm ist auf die Positron–Annihilationslinie von 511 keV hin optimiert. In Tabelle 1.3 sind die Eigenschaften von PICsIT zusammengefasst. Siehe auch [14].

Pixelgröße:	8.55×8.55 mm^2	30 mm Dicke
Pixelabstand:	650 μm	(9.2 mm Abstand zweier Pixelzentren)
Effektive Fläche:	2994 cm^2	($64 \times 64 \times$Pixelgröße)
Energiebereich	15 keV–10 MeV	
Abstand zur Aperturmaske:	3194 mm	94 mm bis zur Oberkante von ISGRI
Zahl der Module:	8	je 512 Pixel

Tabelle 1.3: Schlüsselparameter von PICsIT

Mehrfachereignisse Die Elektronik von PICsIT, welche alle acht Detektormodule verwaltet, kann erkennen, ob mehrere Ereignisse zur gleichen Zeit stattgefunden haben und gibt diese Information an die Datenverarbeitung weiter. Die Energiekorrektur solcher Mehrfachereignisse ist dann relativ einfach indem die Einzelenergien summiert werden.

1.5.5 Kalibration

Zur Kalibration der Detektoren ist eine radioaktive ^{22}Na–Quelle vorgesehen, die sich etwas unterhalb der Aperturmaske befindet. Ihr Spektrum hat zwei Linien bei 511 keV und 1.275 MeV [32]. Diese Strahlenquelle ist mit einer BGO-Abschirmung umgeben, die alle Photonenereignisse dieser Quelle entdecken soll. Tritt ein solches Ereignis auf, dann wird ein Signal an die Elektronik der Detektoren geschickt, die dann ein gleichzeitiges gemessenes Ereignis als Kalibrierungsereignis erkennen und markieren können [32].

1.6 Der Röntgenmonitor (JEM–X)

Der Röntgenmonitor JEM–X (Joint European X–Ray Monitor) besteht aus zwei Microstrip–Detektoren mit kodierter Aperturmaske. Die unter Hochdruck mit Xenon gefüllten Detektoren enthalten eine Streifenplatte und, zur Bestimmung einer Punktkoordinate, darunter eine senkrecht verlaufende Streifenplatte. Das Sichtfeld ist mit Kollimatoren aus Molybdän, die auf dem Detektor sitzen, begrenzt.

1.7 Die optische Monitorkamera (OMC)

Die **Optical Monitoring Camera** (OMC) enthält einem CCD–Chip mit 1024×2048 Pixeln (von denen die Hälfte zur Belichtung, die andere Hälfte zum Auslesen benutzt wird), der auf etwa 193 K gekühlt wird. Vor dem CCD–Chip befindet sich eine Linse mit 50 mm Brennweite. Die Empfindlichkeit ist etwa auf 19.5 mag beschränkt.

Die Datenverarbeitung findet in einer thermisch abgetrennten Elektronik statt, an die wegen der hohen Datenmenge nicht zu vernachlässigende Anforderungen gestellt werden.

1.8 Dithern

Die kodierte Aperturmaske von SPI enthält mehr Felder, als es Detektoren gibt. Daher sollte für bessere Ortsauflösung das Instrument, und damit auch der Satellit, die Strahlungsquelle aus verschiedenen Blickwinkeln betrachten. Aus den Einzelaufnahmen kann man dann ein vollkodiertes Blickfeld zusammensetzen. Diese Methode wird **Dithern**[2] genannt. Die Richtungsänderung des Satelliten wird als **Slew–Mode** bezeichnet. Wenn dagegen der Satellit auf eine Strahlungsquelle ausgerichtet ist, befindet er sich im **Pointing–Mode**.

Dithern bei IBIS Anders als bei SPI kann IBIS wegen der großen Pixelzahl auch ohne Dithern vollständig rekonstruierbare Bilder aufnehmen. Es geht beim Dithern sogar etwas an Beobachtungsinformation verloren, weil IBIS während der Richtungsänderung keine Aufnahmen machen kann, und weil die untersuchte Quelle eventuell aus dem vollkodierten Blickfeld wandert. Vorteilhaft ist das Dithern dennoch auch für IBIS, wenn die Empfindlichkeit der Detektorpixel verschieden ist (oder gar einige Pixel ausgeschalten sind), und ein Gesamtbild zusammengesetzt werden soll.

Dithern bei JEM–X JEM–X ist in der Lage, auch während des Ditherns verwertbare Daten zu liefern, sofern die Richtungsänderung nicht zu schnell stattfindet und jederzeit bekannt ist [24].

[2] Dithern: englisch Schwanken [12].

Kapitel 2

Das Datenverarbeitungssystem von IBIS

Das **Datenverarbeitungssystem** (DHS —Data Handling System) beinhaltet sowohl die Auswertung und Weiterleitung von den Messdaten wie auch die Ausführung von Steuerkommandos. Auf IBIS muss die Datenverarbeitung von drei Detektoren bewerkstelligt werden, nämlich ISGRI und PICsIT, die die wesentlichen Beobachtungsdaten liefern, und das VETO–System, welches für die aktive Abschirmung sorgt.

2.1 Der Datenfluss

Der erste Teil des Datenverarbeitungssystems betrifft die Daten, die von den Detektoren gemessen wurden und nun zur Bodenstation gelangen sollen. Dazu müssen die Daten aus den Detektoren ausgelesen, ausgewertet und komprimiert werden. Ist dies geschehen, so können sie in einzelnen Paketen fester Größe, sogenannten **Telemetriepaketen** (TM) an den Hauptrechner, (Central Data Management Unit — CDMU) übergeben werden, der die Telemetriepakete an die Bodenstation sendet.

Für IBIS beschränkt sich die Datenverarbeitung an Bord des Satelliten im wesentlichen auf die Komprimierung und Registrierung dieser Daten. Es ist aber auch möglich, den Messbereich, z.B. die Energie, einzuschränken und so die Daten zu selektieren. In Bild 2.1 ist der komplette Weg der Datenverarbeitung skizziert.

2.2 Histogramme

Die Datenrate an Einzelereignissen, die von ISGRI und PICsIT erwartet werden, ist mit etwa 2000 cts[1] (ISGRI) bzw. 7500-15000 cts[2] (PICsIT) sehr hoch. Es

[1]Aus [11, Seite 18].
[2]Aus [14, Seite 276].

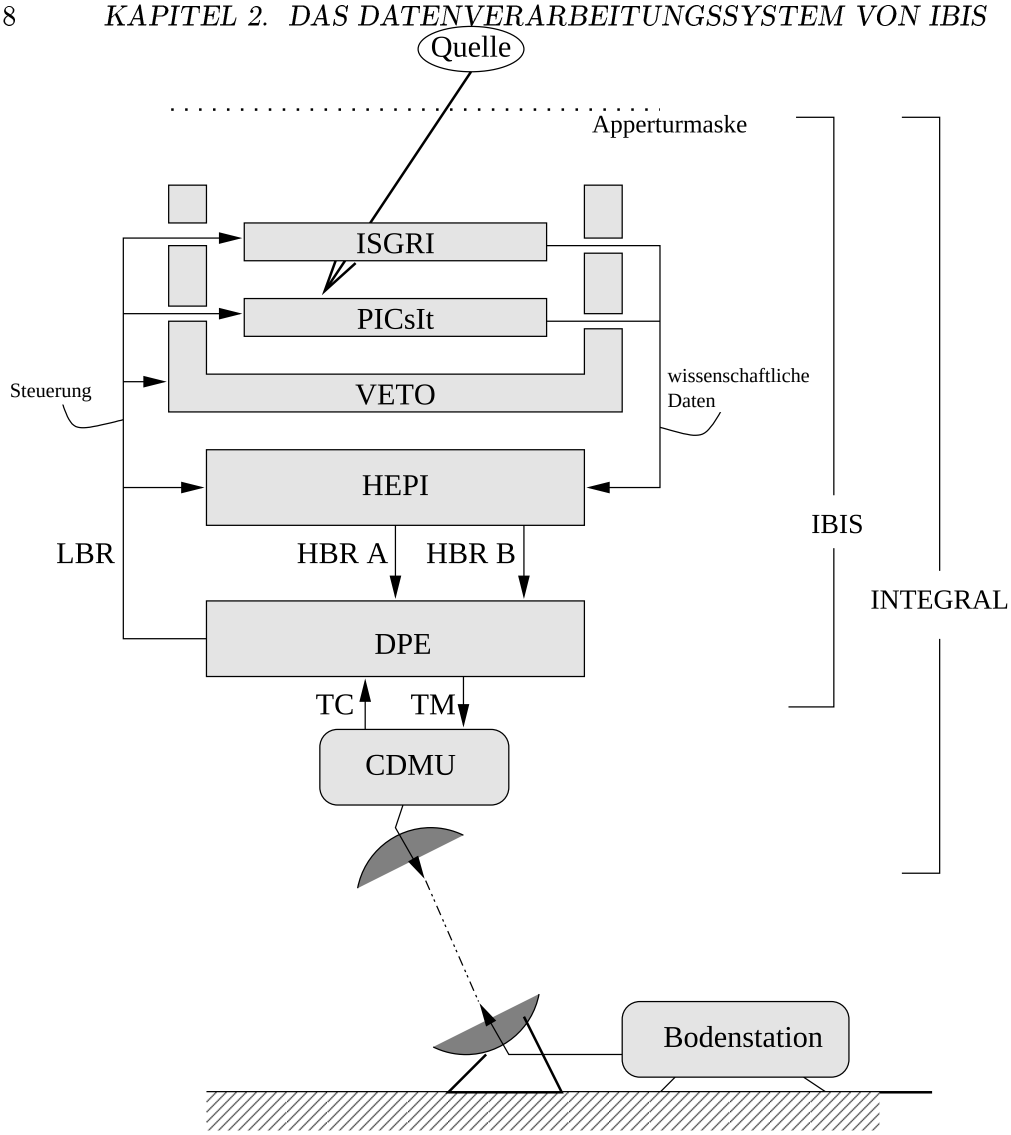

Abbildung 2.1: Schema der Datenverarbeitung

gibt verschiedene Strategien, um diese Informationsmenge möglichst verlustlos zu reduzieren. Ein Ansatz besteht darin, im Parameterraum der Messergebnisse Werte zusammenzufassen (dies entspricht einer Integration). Häufig werden über eine Integrationszeit Einzelereignisse in verschiedenen Zellen gezählt, die ein sogenanntes **Histogramm** bilden.

Für das IBIS–Instrument ist primär die Ortsinformation der Detektorereignisse von Bedeutung. Dazu sind Histogramme gut geeignet, weil auch bei hohen Datenraten die Ortsverteilung erhalten bleibt. Die Zeitinformation wird dabei stark reduziert.

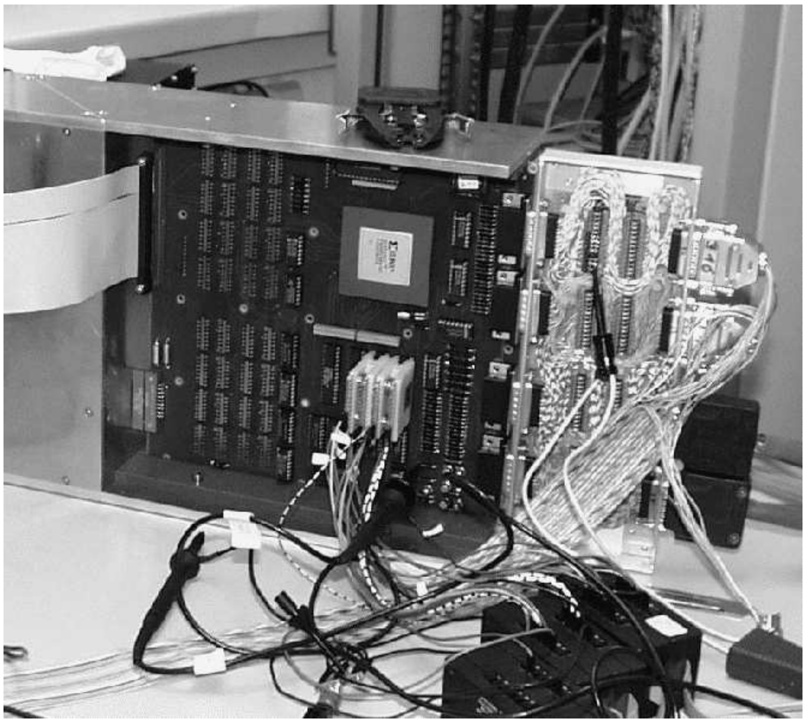

Abbildung 2.2: HEPI–Bord im Testrahmen

2.3 Der Vorprozessor: HEPI

Wegen der hohen Datenrate benötigt man auf IBIS ein schnelles Datenverarbeitungsgerät. Die gängigen Mikroprozessoren (und besonders die, die raumfahrttauglich sind) sind dazu nur eingeschränkt in der Lage, da sie zu langsam sind. Daher muss eine schnelle Elektronik einen wesentlichen Teil der Datenverarbeitung übernehmen.

Diese Elektronik, die sich **HEPI** (Hardware Event Preprocessor for IBIS) nennt, führt eine Vorverarbeitung der Daten durch und kann zur Datenreduktion verschiedene Funktionen benutzen:

- Die Zeitinformation der Detektordaten (die sich auf einen 2–Sekunden–Takt beziehen) muss für eine sinnvolle Messung auf einen absoluten Zeitpunkt bezogen werden.
- Mehrfachereignisse von PICsIT werden zusammengefasst.
- Gleichzeitige Ereignisse von ISGRI und PICsIT werden zu Compton–Ereignissen zusammengefasst.
- Kalibrationsereignisse werden markiert.
- Die Ortsinformation kann verworfen werden, um eine periodische Energie/Zeit–Information zu erhalten (Spectral Timing). Dazu wer-

den Energiehistogramme über Zeiten von 1 msec bis 500 msec in acht Kanälen integriert.

- Für eine Datenreduktion von PICsIT -Ereignissen können Histogramme gebildet werden. Dazu werden über einen Zeitraum von typischerweise 1000 Sekunden die Einzelereignisse, nach Ort- und Energieinformation sortiert, in die einzelnen Zellen gezählt. Die Energie wird mit 256 Kanälen aufgelöst.
- Ein besonderes Histogramm bestimmt aus PICsIT–Mehrfachereignissen eine Polarisationsinformation, indem die Richtung benachbarter Pixel–Ereignisse ausgewertet wird. Dies beruht darauf, dass Mehrfachereignisse von Gammaquanten bevorzugt in Richtung des **E**-Vektors auftreten (Compton–Streuung).
- PICsIT–Kalibrationsereignisse können in einem Histogramm mit 64 Energiekanälen, 32 pro Linie, integriert werden.

Diese Funktionen sind kombinierbar. Zudem ist es möglich, die HEPI[3] transparent zu schalten, d.h. alle Einzelereignisse für Diagnosezwecke unverändert weiterzuleiten. In Bild 2.3 ist schematisch dargestellt, wie der Datenfluss der HEPI aussieht. Für genauere Information siehe [13, 35, 34].

Statusregister Um die Funktionen der HEPI an– oder auszuschalten, gibt es ein **Statusregister**. Die Funktionen sind durch Bits repräsentiert, die in Tabelle 2.1 zusammengestellt sind.

Energieaufteilung bei Histogrammen Bei allen Histogrammen in der HEPI ist die Zahl der Energiekanäle (256 bzw. 64 für Einzelereignis/Kalibrationsereignis) kleiner als der Wertebereich der Energiedaten, die von den Detektoren kommen (1024 Kanäle für PICsIT). Um die Energieauflösung zu erhöhen lässt sich die Zuordung zwischen gemessener Energie und Kanal im Histogramm mit einer Tabelle (LUT — Look up Table) beliebig konfigurieren. Dadurch ist es z.B. möglich, einzelne Linien im Gamma–Spektrum höher als das Kontinuum aufzulösen.

2.4 Die Data Processing Electronic (DPE)

Jedes Instrument hat zur Datenverarbeitung einen Computer, die sogenannte **Data Processing Electronic** (DPE). Für SPI, JEM–X und IBIS sind diese aus Redundanzgründen doppelt vorhanden, von denen normalerweise nur die **nominale DPE** verwendet wird. Diese DPE liest von HEPI die Messdaten aus, verpackt sie und schickt sie in Telemetriepaketen an den Hauptrechner von INTEGRAL, die CDMU, weiter. Aus Kostengründen verwenden alle Instrumente auf INTEGRAL eine baugleiche DPE.

[3] Das grammatikalische Geschlecht von HEPI wird unterschiedlich behandelt[13, 27]. In der vorliegenden Arbeit wird HEPI feminin verwendet, da dies dem allgemeinen Sprachgebrauch am ehesten entspricht. Vgl. auch [6, Seite 207]

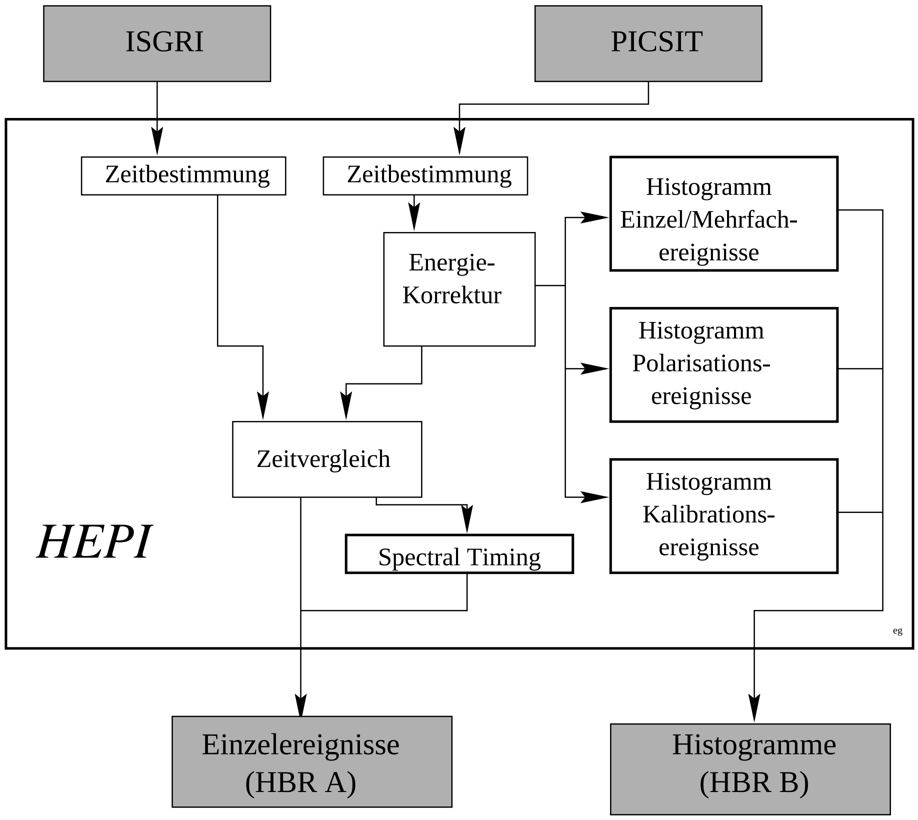

Abbildung 2.3: Schematischer Datenfluss in der HEPI

2.4.1 Aufbau der DPE

Prozessor Die DPE enthält einen Prozessor des Types 1750A[4], der unter anderem im Satellitenbereich breite Anwendung gefunden hat. Er hat eine 16-Bit CISC-Architektur, d.h. die Maschinenbefehle sind auf 16 Bit–Daten ausgelegt und mit denen eines 8086–Prozessors vergleichbar. Vorteilhaft sind die standardmäßig integrierten Fließkommabefehle. Die Rechenleistung liegt mit 13 Mhz Taktfrequenz bei etwa einer Millionen Instruktionen pro Sekunde (1 MIPS).

Speicherarchitektur Die Adressierung des Speichers geschieht indirekt:

1. **Logische Adressen**: Der Prozessor kann auf einen Adressraum von 64 kWord Daten zugreifen. Die Maschinenbefehle sind in einem separaten, ebenfalls 64 kWord großen Speicherbereich untergebracht.

2. **Paging**: Diese logischen Adressen werden in 4 kWord–Blöcken (»Pages«) dem **physikalischen Speicher** zugeordnet. Es gibt folglich 16 Pages für den logischen Adressraum.

[4] Dieser Typ ist eine Prozessorspezifikation, die vom amerikanischen Department of Defense entwickelt wurde. Er ist von verschiedenen Firmen implementiert worden.

Bits	Beschreibung der Funktion
0	ISGRI auslesen (invertiert)
1	PICsIT auslesen (invertiert)
2	PICsIT–Energiekorrektur
3	Zählratenbestimmung
4	Rekonstruktion von Mehrfachereignissen
5	Rekonstruktion von Polarisationsereignissen
6	Energie–Zeit–Histogramme (Spectral Timing)
7	PICsIT–Energiebeschränkung, Einzelereignisse
9	Kalibrationshistogramm
10	Einzel/Mehrfachereignis-Histogramm
12	Polarisationshistogramm
13	Zeitvergleich
16	ISGRI–Einzelereignisse
19	Science–Histogrammbereich (1/2)
20	Kalibrationhistogrammbereich (1/2)

Tabelle 2.1: Aufbau des HEPI Status–Registers

3. **Adresszustände**: Die Zuordnung logischer Adressraum–physikalischer Adressraum über Pages kann gewechselt werden. Jede Zuordnung ist ein **Adresszustand** (englisch: Address State). Es kann bis zu 16 Adresszustände geben, zwischen denen gewechselt werden kann.[5]

Der physikalische Speicher der DPE ist 1 MWord (2^{20} Word) groß.

E^{2}PROMs Das vollständige Programm der DPE befindet sich in 20 programmierbaren Speicherbausteinen (EEPROM — Electric Erasable Programmable Read Only Memory) mit je 8 kByte Speichervermögen, die auf einem speziellen Bord der DPE untergebracht sind. Bei der Initialisierung der DPE wird der Inhalt dieser PROMs in den physikalischen Speicher kopiert und das Programm gestartet.

Die Änderung eines Programms der DPE gestaltet sich daher etwas umständlich, weil zunächst das Bord aus der DPE gezogen und geöffnet werden muss, worauf die PROMs zugänglich sind und einzeln neu programmiert werden können.

Schnittstellen zu den Detektoren Die Datenverbindung zur HEPI besteht aus zwei schnellen, seriellen Leitungen, (**High speed Serial Line** — HSL). Eine andere Bezeichnung für diese Leitung ist die High Bit Rate Line (HBR). Die eine Leitung (HBR A) dient zur Übertragung von Einzelereignissen, die andere (HBR B) zum Auslesen von Histogrammen. Die Übertragungsgeschwindigkeit ist 5 Mbps. Die Übertragung verläuft synchron.

[5]Die Zuordnung, die in einem Prozessorregister festgehalten ist, kann auch umprogrammiert werden. In diesem Falle sind mehr als 16 Adresszustände möglich; dieser Eingriff ist jedoch nicht trivial.

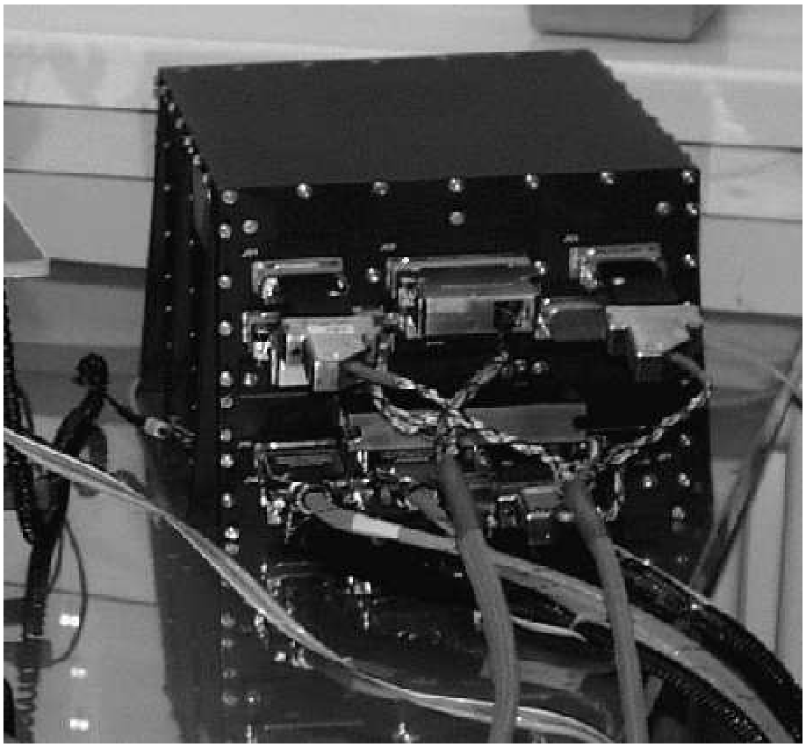

Abbildung 2.4: Die DPE mit ihren Anschlussleitungen

Zu allen Detektoren (ISGRI, PICSIT und VETO) und zur HEPI gibt es zusätzlich eine serielle Steuerleitung mit einer Übertragungskapazität von 16 kbps asynchron. Sie wird als **Low speed Serial Line** (LSL) oder Low Bit Rate Line (LBR) bezeichnet. Über diese Leitung können entweder Kommandos abgeschickt oder Parameter ausgelesen werden. Es gibt von der DPE einige weitere Leitungen zum Auslesen von Analogwerten und um Spannungspulse an die Detektoren schicken zu können.

2.5 Verbindung zur CDMU

2.5.1 Telemetrie

Die Daten werden in der DPE in einzelne, unabhängige Datenpakete (**Telemetrie** — TM) verpackt. Diese gelangen über eine spezielle Verbindung, dem **On–Board Data Handling Bus** (OBDH–Bus) zum Hauptrechner von INTEGRAL (CDMU), der sie zur Bodenstation sendet.

Klassifizierung der Telemetriepakete Jedes Telemetriepaket muss eindeutig identifizierbar sein. Dazu wurde von der Firma Alenia eine Struktur

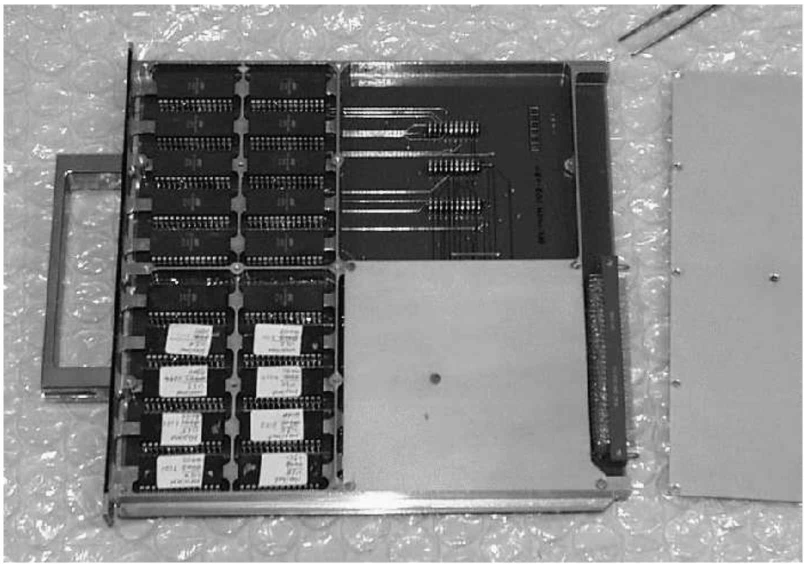

Abbildung 2.5: DPE–PROM–Bord bei der Bestückung

entwickelt, die für den ganzen Satelliten verbindlich ist.[6] Diese Struktur ordnet jeder DPE mehrere Nummern zu, die **APID**s (Application Identifier) genannt werden. Mit ihnen kann der Sender von Telemetriepaketen festgelegt werden. Für die IBIS–IASW sind APIDs im Bereich von 1281 bis 1407 vorgesehen (nominale DPE)[7].

Telemetriepakete werden weiter in **Typen** und **Subtypen** unterteilt, wobei jedem (Typ, Subtyp)–Paar eine bestimmte Aufgabe zugewiesen ist [22]. Daher werden sie häufig mit TM (X,Y) abgekürzt, wobei X,Y den Typ und Subtyp angibt. Alle Telemetriepakete mit einer APID zwischen 1281 und 1343 sind für Steuerungsaufgaben mit fest definierten Aufgaben vorgesehen, während die übrigen frei für wissenschaftliche Daten verwendet werden können. In dieser Arbeit sind vor allem die Telemetriepakete, die bei Steuerungsaufgaben anfallen, von Bedeutung. Sie sind in Tabelle 2.2 aufgeführt. Die wissenschaftlichen Telemetriepakete sind in [27] beschrieben.

Struktur der Telemetriepakete Jedes Telemetriepaket besteht aus einem Kopf, der drei Word lang ist, die dem unter anderem die APID enthält. Im vierten Word sind Typ und Subtyp abgelegt. Das fünfte Word enthält die Zeit, zu der das Telemetriepaket abgeschickt wurde.[8] Nach diesem Word folgen die Daten, die mit dem Telemetriepaket übermittelt werden sollen. Für eine Parameterunterscheidung sind bei dem Telemetriepaket »Report Parameter« TM(5,4) zwei weitere Bezeichner, nämlich die **Task–ID** (TID) und **Function–ID** (FID)

[6] siehe: Integral Packet Structure Definition [22].

[7] Für die CSSW ist die APID 1280 vorgesehen.

[8] Die Einheit ist üblicherweise in Sekunden der An–Bord–Zeit, kann aber auch anders definiert sein [27].

APID	Typ	Subtyp	Beschreibung
1281	1	8	periodischer Statusbericht (Housekeeping–Paket).
1281	1	1	Statusbericht wissenschaftlicher Datenerzeugung (Science Housekeeping–Paket).
1281	1	10,11	Zusätzlicher Statusbericht von VETO (VETO–Spectra). Besteht aus zwei Paketen.
1281	1	2-9	Zusätzlicher Statusbericht von ISGRI und PICsIT. Besteht aus acht Paketen, von denen je eines einem ISGRI–MCE zugeordnet ist.
1281	5	4	Parameter, die ausgelesen wurden.
1290	9	1	Statusbericht, welche Telemetriepakete erzeugt werden.

Tabelle 2.2: Verschiedene Telemetriepakete

vorangestellt, die den Parameter nach Aufgabenbereich und Funktion einordnen. Die gesamte Struktur ist in Tabelle 2.3 aufgelistet. Die Bitbelegung eines Telemetriepaketes ist in Tabelle 2.4 dargestellt.

Bezeichner	Beschreibung
APID	Application Identifier
Sequence Counter	Telemetrie–Zähler
Packet Length	Länge des Telemetrie–Paketes (gesamte Länge minus 7, in Byte). Die totale Länge ist 220 Word.
Type	Typ des Telemetriepaketes.
Subtype	Subtyp des Telemetriepaketes
Zeit	Zeit des Abschickens.
TID	Task–Identifier (kann fehlen)
FID	Function–Identifier (kann fehlen)
Data	bis zu 215 Word Daten

Tabelle 2.3: Inhalt eines Telemetriepakets

2.5.2 Telekommandos

Zum Steuern des Instruments kann die Bodenstation **Telekommandos** (**TC**) an den Satelliten schicken, die von der CDMU zur DPE übermittelt werden. Die Kommandos können über die LSL an die Detektoren von IBIS und zur HEPI weitergeschickt werden.

Klassifizierung der Telekommandos Jedes Telekommando muss eindeutig identifizierbar sein, so dass sie ebenfalls eine einzelne APID verwenden. Für Telekommandos genügt eine APID, die bei IBIS 1281 (nominale DPE) ist. Jedem **Typ** und **Subtyp** ist auch bei Telekommandos eine bestimmte Aufgabe zuge-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4			0	1	APID										

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1		Sequence–Counter													

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Länge des Telemetriepaketes (433 Byte)															

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Zeit beim Abschicken (sec)															

64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Typ				Subtyp											

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
TID								FID							
Word 5..N: Parameter															

Tabelle 2.4: Struktur eines Telemetriepaketes

wiesen ist. In Tabelle 2.5 sind die verschiedenen Paare, die in der IBIS–Software Anwendung finden, aufgeführt. Sie werden analog zu den Telemetriepaketen mit TC(X,Y) abgekürzt, wobei X,Y den Typ und Subtyp bezeichnet.

Das erste (Typ, Subtyp)–Paar, auch mit TC(2,1) bezeichnet, bezeichnet Telekommandos, die direkt auf die Hardware wirken können, indem ein elektrisches, pulsförmiges Signal auf eine Ausgangsleitung der DPE gelegt wird. So ein **Puls** kann z.B. ein Zurücksetzen eines Instrumententeils bewirken.

Um Steueraufgaben zu bewältigen, gibt es mehrere Telekommandos (die ≫5–er Serie≪), die für verschiedene Aufgaben oder Tasks[9] verwendet werden können. Für das Konzept von „Arm and Fire“ werden für eine bestimmte Funktion, die das Instrumente zu erfüllen hat, zunächst die Konfigurationsparameter geladen, woraufhin die Funktion aktiviert oder auch deaktiviert werden kann. Für die Verifikation muss zudem jeder Parameter auslesbar sein.

Jeder Funktion des Instrumentes wird ein **Task–Identifier** (TID) zugeordnet, der in verschiedene (Unter–)Funktionen (**Function–Identifier**, FID) aufgeteilt werden kann. Jedem Satz von Konfigurationsparametern einer bestimmten Funktion ist ein (TID/FID)–Paar zugeordnet.[10] Die Struktur eines solchen Parametersatzes muss konstant sein.

Für die allgemeine Funktionalität der Software ist es von Vorteil, wenn die

[9]Diese Tasks sind nicht die Softwaretasks, die für das Multitasking verwendet werden, sondern beschreiben nur die allgemeinen Funktionen, die für die Steuerung nötig sind.

[10]In [22] wird festgelegt, dass jeder einzelne Parameter einer eigenen TID–FID–Kombination zugeordnet ist. Für IBIS ist dieser Parameterraum zu klein. Daher ist einer TID–FID–Kombination ein *Parametersatz*, der auch mehrere Parameter enthalten kann, zugeordnet.

Typ	Subtyp	Name	Beschreibung
2	1	Puls–Kommando	Schickt einen Puls auf eine bestimmte Leitung.
5	1	Starte Task	Beginnt mit dem Ausführen einer bestimmten Aufgabe.
5	2	Stoppe Task	Beendet die Ausführung einer bestimmten Aufgabe.
5	3	Lade Task Parameter	Setzen eines Parameters oder Parametersatzes für eine bestimmte Aufgabe. Damit können sowohl die Software und auch die anderen Teile des Instruments konfiguriert werden.
5	4	Lese Task Parameter	Liest einen Parameter aus und schickt den Wert in einem Telemetriepaket zur Bodenstation.
5	5	Modus ändern	Ändert in der Software den Arbeitszustand.
9	1	Lese Telemetrietatus	Liest den Zustand der Generierung von Telemetriepaketen aus und schickt ihn als Telemetriepaket an die Bodenstation.
9	4	Schalte Telemetrie an	Ermöglicht die Erzeugung bestimmter Telemetriepakete.
9	5	Schalte Telemetrie aus	Verhindert die Erzeugung bestimmter Telemetriepakete.
15	1	Broadcast–Paket	Enthält Satelliteninformationen, die alle acht Sekunden an jede DPE von INTEGRAL gesendet werden.

Tabelle 2.5: Typ–Subtyp–Belegung

verschiedenen Arbeitsvorgänge in **Modi** unterteilt werden, die sich gegenseitig ausschließen. Dadurch kann vermieden werden, dass die Software zu viele, evtl. sich wechselseitig beeinflussende Dinge zu erledigen hat, die zu einem unstabilen Verhalten, wenn nicht gar zum totalen Ausfall führen können. Zum Umschalten zwischen diesen Modi ist ein Telekommando vorgesehen, welches quasi als Hauptschalter des Instruments dient.

Zusätzlich gibt es noch Telekommandos, mit denen die Erzeugung von bestimmten Telemetriepaketen an– oder ausgeschalten werden kann. Das ist besonders für Diagnosezwecke nützlich, wenn nur bestimmte Telemetriepakete von Belang sind und andere ausgeblendet werden sollten, um die begrenzte Übertragungsbandbreite von INTEGRAL voll auszunutzen.

Bezeichner	Beschreibung
APID	Application Identifier
Sequence Counter	Telekommando–Zähler; oberstes Bit bezeichnet die Herkunft des Telekommandos (an Bord/Bodenstation)
Packet Length	Länge des Telekommand–Paketes (gesamte Länge minus 7, in Byte)
CRC–Flag	Existenz einer Prüfsumme
Type	Typ des Telekommandos
Subtype	Subtyp des Telekommandos
TID	Task–Identifier (kann fehlen)
FID	Function–Identifier (kann fehlen)
Data	bis zu 119 Word Daten/Parameter des Telekommandos (Länge je nach Parametergröße)
CRC	Prüfsumme (kann fehlen)

Tabelle 2.6: Inhalt eines Telekommandos

Struktur der Telekommandos Jedes Telekommando besteht aus einem Kopf, der drei Word von je 16 Bit lang ist. In diesem befindet sich die APID, ein Zähler für die Nummerierung der Telekommandos (Sequence Counter), der für die Kontrolle der ausgeführten Kommandos sehr wichtig ist, und diverse Felder, die dem Paket die Eigenschaft ≫Telekommando von INTEGRAL ≪ zuweisen.

Danach kommen ein Word, das neben dem (Typ, Subtyp)–Paar ein Pfrüfsummenflag enthält. Ist es gesetzt, so ist dem Telekommando eine Prüfsumme angehängt. Diese Prüfsumme (auch CRC— Cyclical Redundancy Check) dient einer sicheren Übertragung von **kritischen Telekommandos**.

Nach dem (Typ, Subtyp)–Feld kann noch die Task–ID (TID) und Function–ID (FID) folgen, sofern es sich um ein Task–spezifisches Telekommando handelt. Die relevanten Felder eines Telekommandos sind in Tabelle 2.6, ihre Bitbelegung in Tabelle 2.7 dargestellt.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0			1	1	APID										

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
3		Sequence–Counter													

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Länge des Telekommandos															

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0		CRC–Flag		0				Typ				Subtyp			

64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
TID								FID							
Word 5..N: Parameter															

Tabelle 2.7: Struktur eines Telekommandos

2.5.3 Polling Sequence

Alle Instrumente kommunizieren über denselben OBDH–Bus mit der CDMU. Für eine flexible Telemetrieratenvergabe gibt es eine **Polling–Sequence–Table** (PST), die für einen **Polling–Zyklus** von acht Sekunden regelt, wann welches Instrument Telemetriepakete bzw. Telekommandos senden bzw. erhalten kann. Die Einträge der Polling-Sequenz–Tabelle sind in Fenster von 1/46 Sekunden aufgeteilt. Jedes Instrument kann höchstens jedes zweite Fenster ein Telemetriepaket schicken, woraus sich eine Rate von maximal 23 Telemetriepaketen pro Sekunde ergibt.

2.5.4 An–Bord–Zeit

Die CDMU ist für eine definierte Zeit (**On–Board–Time** — OBT) in INTEGRAL verantwortlich, die sie über ein spezielles Synchronisationsprotokoll an die DPE jedes Instruments schickt. Diese Zeit kann, wie in Tabelle 2.8 abgebildet, in 43 Bit mit einer Auflösung von 1.9 μsec ($2^{-19} sec$) dargestellt werden.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MSW der Zeit (sec)															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Zeit in sec								Zeit in 2^{-19} sec, MSB							

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Zeit in 2^{-19} sec											X	X	X	X	X

Tabelle 2.8: Struktur der An–Bord–Zeit (OBT)

Kapitel 3

Die IASW und ihre Aufgabe

In der DPE gibt es ein Programm, welches die Messdaten verarbeitet und die Detektoren sowie die HEPI steuert. Diese Software, die das zentrale Thema der vorliegenden Arbeit ist, ist die Verbindung zwischen der (CDMU) und den IBIS–Detektoren, welche durch die bereits vorgestellten Elektronik–Einheiten kontrolliert werden. Sie wird daher als »**Instrument Application Software**« (kurz **IASW**) bezeichnet, weil sie spezifisch auf das IBIS–Instrument zugeschnitten ist. Da es für die anderen Instrumente (SPI, JEM–X und OMC) ebenfalls eine entsprechende Software gibt, wurde ein Systemkern entwickelt, der der IASW möglichst viele allgemeine Aufgaben abnimmt. Dieser Softwareteil, der als Betriebssystem arbeitet, wird folglich **CSSW** (Common Service Software) genannt und ist von der Firma GMV, Madrid entwickelt worden.

In diesem Kapitel werden zunächst die Anforderungen an die Software (eine Kurzform der Spezifikation) umrissen. Die Aufgaben der Software lassen sich grob in drei Bereiche unterteilen:

- Steuern des Instrumentes
- Auslesen der Betriebsparameter (Housekeeping–Pakete)
- Auslesen und Verarbeiten von wissenschaftlichen Daten

Die Verbindung ist in Bild 3.1 dargestellt.

3.1 Steuerung des IBIS–Instruments

Jeder Detektor oder Prozessor des IBIS–Instruments ist über eine serielle Schnittstelle oder über verschiedene Digital– und Analogkanäle von der DPE aus erreichbar. Daher arbeitet die Software in der DPE als eine Zentrale, die die Steuerung über das ganze Instrument übernimmt.

Die Steuerung des Instrumentes erfolgt entweder durch Telekommandos, die von der Bodenstation über die CDMU die IASW erreichen und von ihr weitergeschickt werden, oder durch die IASW selber, die Steuerbefehle an die verschiedenen Instrumentteile schicken kann. Letzteres ist entweder die Ausführung eines komplexen Protokolls (Housekeeping-Daten oder Histogrammbearbeitung

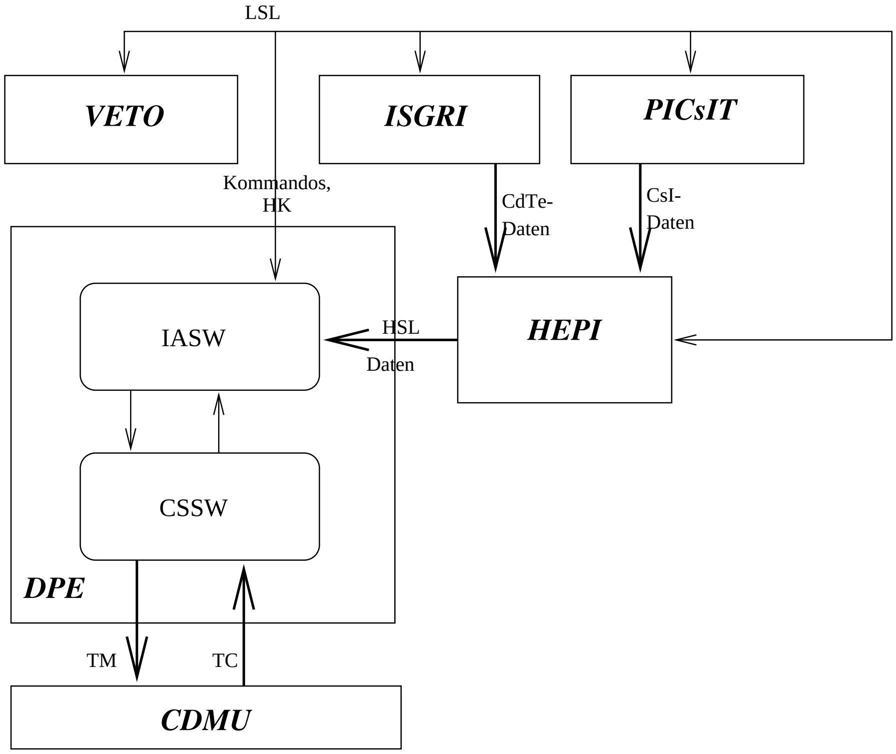

Abbildung 3.1: Daten/Steuerfluss in IBIS

gehört dazu), oder die IASW reagiert autonom auf eine Änderung des Satellitenzustandes (etwa Abschalten der Hochspannung bei erhöhter Strahlung).

3.1.1 Telekommandos

Bevor die internen Abläufe der IASW beschrieben werden, soll zunächst dargestellt werden, welche Telekommandos die IASW entgegennimmt und wie sie auf sie reagiert.

Externe Telekommandos

Wenn das Telekommando nicht die IASW/CSSW betrifft, sondern an einen anderen Teil des Instruments (ein **Subinstrument**) geschickt werden soll, so wird es von der IASW unbesehen weitergeleitet. Die vier Subinstrumente von IBIS sind ISGRI, PICsIT, VETO und HEPI. Diese Kommandos werden als **externe Kommandos** bezeichnet.

Ein externes Telekommando enthält neben dem Instument–Befehl und seiner Länge den Empfänger (nämlich eines der vier Subinstrumente) und die Länge einer Antwort von dem Subinstrument, da die DPE die volle LSL–Übertragung steuert. Ist die Antwort–Länge Null, handelt es sich um ein Kommando, von dem keine Antwort erwartet wird. Der Vorteil ist, dass die IASW nichts über die Bedeutung des Kommandos wissen muss, so dass es transparent ohne

Interpretation weitergeschickt werden kann.[1] Dies bedeutet allerdings, dass die Bodenstation die Konsistenzprüfung von externen Telekommandos übernehmen muss.

Von der Logik der Telekommando–Struktur, die von der Firma Alenia festgelegt wurde, sollten externe Telekommandos vom Typ »Start Task« bzw. »Report Task Parameter« (wenn eine Antwort erwartet wird) sein. Aber nur das Telekommando »Lade Task Parameter« (TC(5,3)) kann alle Informationen als Parameter in einer größeren, wenn auch konstanten Struktur aufnehmen, so dass dieser Telekommdotyp für externe Telekommandos verwendet wurde.[2]

Struktur externer Telekommandos Jedem Subinstrument ist eine Adresse, die sogenannte **Subinstrument Address** zugeordnet [33].

Die Task–ID eines externen Telekommandos dient generell zur Unterscheidung der vier Subinstrumente, an die der Befehl gehen soll, und die Function–ID legt den Befehl selber fest. Allerdings ist diese Festlegung nur für die Verwaltung der Telekommandos gedacht. Die IASW überprüft keine Konsistenz zwischen der Task–ID und der Subinstrument–Address[3], d.h. die Funktion–ID wird von ihr nicht berücksichtigt.

Die ersten Parameter legen die Länge der Antwort eines Subinstrumentes (in Words) und die Subinstrument–Address fest. Dann folgt der Befehlscode variabler Länge, den das Subinstrument erhalten soll. Der Aufbau externer Telekommandos ist in Tabelle 3.1 dargestellt. Für die genaue Definition siehe [33].

Word 1-5: Kopf des Telekommandos wie in Tabelle 2.7															
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Länge der Rückantwort								Subinstrument–Adresse							
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
Länge des Kommandos								Kommando–Identifier (CID)							
Weitere Parameter können folgen															

Tabelle 3.1: Struktur eines externen Telekommandos

[1] Die Probleme, die sich durch die internationale Kooperation ergeben können, sollte man auch im Design nicht vernachlässigen. Es hatte sich für das Testen des Systems als sehr sinnvoll erwiesen, nur durch Ändern von Telekommandos den unterschiedlichen Konfigurationen der Subsysteme Rechnung zu tragen.

[2] Von einem gewissen Standpunkt aus kann man externe Telekommandos tatsächlich als solche ansehen, die einen Parameter setzen — und sei es nur der, dass ein Flag »Gerät an/abschalten« gesetzt wird. Dennoch muss zugegeben werden, dass hier das Konzept von Tasks, wie es in [22] beschrieben ist, nicht voll eingehalten ist.

[3] Von der IASW wird immer nur die Sub–Instrument–Address berücksichtigt. Im Gegensatz zur FID kann sie zwischen den acht Modulen des ISGRI–Instruments unterscheiden.

Interne Telekommandos

Zu den **internen Telekommandos**, die direkt in der IASW ausgeführt werden, zählen solche, die den Softwaremodus wechseln, Parameter innerhalb der IASW verwalten und Statusberichte anfordern. Auch Telekommandos, die eine umfangreiche Transaktion zwischen der DPE und den Subinstrumenten starten (etwa Histogramme auslesen oder die Instrument–Konfiguration sichern) werden zu den internen Telekommandos gerechnet, weil sie eine Aktion der IASW anstoßen.

Die Modi der IASW Ein besonders wichtiges Konzept ist die schon erwähnte Aufteilung der Softwareaufgaben in mehrere Modi. Für die IASW gibt es drei Hauptmodi, die sich noch in weitere Untermoden separieren können. Diese sind:

- Der **Standby–Mode** ist für alle Konfigurierungsaufgaben der IASW und der Subinstrumente gedacht. In ihm findet keine wissenschaftliche Datenverarbeitung statt.

- Der **Science–Mode** liest wissenschaftliche Daten ein, verarbeitet sie gemäß der Konfigurierung und schickt sie an die Bodenstation. Hier kann nur ein Moduswechsel–Telekommando geschickt werden.[4]

- Der **Diagnostic–Mode** dient der Kalibration und Fehlersuche. Er hat die größte Freiheit, da sowohl wissenschaftliche Daten eingelesen werden wie auch sämtliche Telekommandos erlaubt sind.

Der Science–Mode kann in drei verschiedenen Konfigurationen, die die wissenschaftlichen Funktionen von HEPI aktivieren, arbeiten. Diese werden **Submode** genannt. Da der Satellit entweder stabil in eine bestimmte Richtung blickt (Pointing–Mode) oder die Richtung wechselt (Slew Mode), muss der Submode entsprechend verschiedene Funktionen aktivieren. In Bild 3.2 sind die verschiedenen erlaubten Zustandsübergänge dargestellt. Die Modi der IASW können mit einem Telekommando vom Typ »Modus ändern« (TC(5,5) gewechselt werden. Außerdem kann die Point–Slew–Konfigurierung des Science Mode mittels eines zusätzlichen Parameters festgelegt werden (sofern der noch zu behandelnde Point–Slew–Automatismus aus ist). (Siehe Tabelle 3.2.)

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
Modus								Point–Slew–Konfiguration							

Tabelle 3.2: Parameter für ein Telekommando »Modus ändern«

[4] Wird ein anderes Telekommando geschickt, so wird es nicht ausgeführt, sondern mit einer Fehlermeldung quittiert, denn die volle Leistung der DPE wird für die Datenverarbeitung gebraucht. Zudem sollte während einer laufenden Messung im Science–Mode die Konfiguration nicht veränderbar sein.

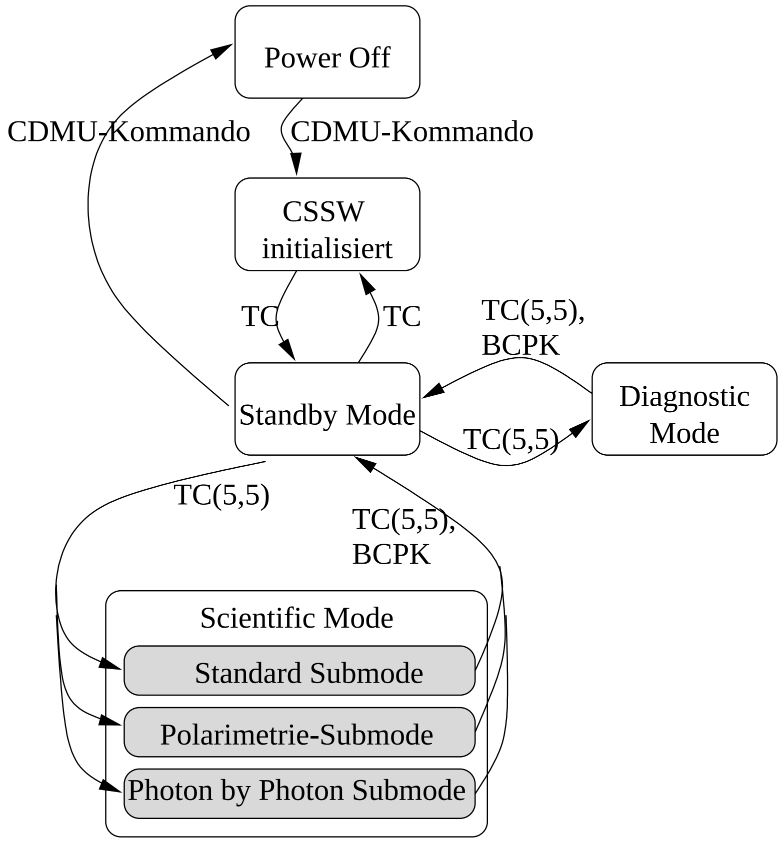

Abbildung 3.2: Die verschiedenen Modi von IBIS

Die Parameter der IASW Es gibt verschiedene Arten von Parametern:

- Boolsche Werte oder Flags (zum Ein/Ausschalten von gewissen Funktionen)
- Schwellwerte
- Spezialparameter
- Parameterblöcke, die **Parametertabellen** genannt werden.

Die Flags dienen vor allem dazu, um sogenannte **Automatismen** ein– und auszuschalten. Automatismen sind komplexe Steuerungsvorgänge, die von der IASW unter bestimmten Umständen vorgenommen werden und die auf alle Fälle abschaltbar sein müssen. Eine weitere Form ist in dem sogenannten **Electronics Configuration Register** (ECR) vorhanden. In ihm ist festgehalten, welches Subinstrument überhaupt angeschaltet und benutzbar ist. **Schwellwerte** bestimmen, welcher Bereich einer Messvariable (meistens die Amplitude bzw. die davon abgeleitete Energie, unter Umständen aber auch die Zählrate) für eine Messung oder eine Steuerung berücksichtigt werden sollen. Ein Spezialparameter betrifft ein Konfigurationsregister, nämlich das **Register Typ II** der HEPI, das beim Start einer Messung zur HEPI übertragen wird. Es gleicht aber sonst den Schwellwert–Parametern. Für jeden dieser Parameter gibt es ein Telekommando vom Typ »Lade Task Parameter« mit einer bestimmten Task/Function–ID.

Es gibt aber auch ganze Sätze von Parametern, die unterstützt werden müssen. Um nun nicht für jeden einzelnen Parameter eine Task/Function–ID verwenden zu müssen, werden diese Parametersätze in **Parametertabellen** zusammengefasst. Diese Tabellen haben ein einheitliches Format, das in Tabelle 3.4 dargestellt ist. Jede Parametertabelle hat eine eindeutige Nummer und ist in **Parameter–Blöcke** von 128 Byte unterteilt. Die Blöcke sind von eins beginnend nummeriert und können einzeln mit einem Telekommando vom Typ »Lade Task Parameter« verändert werden, dessen Struktur in Tabelle 3.3 dargestellt ist. Der Inhalt eines Tabellenblocks kann verschieden strukturiert sein.

<table>
<tr><td>64</td><td>65</td><td>66</td><td>67</td><td>68</td><td>69</td><td>70</td><td>71</td><td>72</td><td>73</td><td>74</td><td>75</td><td>76</td><td>77</td><td>78</td><td>79</td></tr>
<tr><td colspan="8">TID</td><td colspan="8">FID</td></tr>
</table>

<table>
<tr><td>72</td><td>73</td><td>74</td><td>75</td><td>76</td><td>77</td><td>78</td><td>79</td><td>80</td><td>81</td><td>82</td><td>83</td><td>84</td><td>85</td><td>86</td><td>87</td></tr>
<tr><td colspan="16">Blocknummer</td></tr>
</table>

<table>
<tr><td>80</td><td>81</td><td>82</td><td>83</td><td>84</td><td>85</td><td>86</td><td>87</td><td>88</td><td>89</td><td>90</td><td>91</td><td>92</td><td>93</td><td>94</td><td>95</td></tr>
<tr><td colspan="16">1. Parametereintrag</td></tr>
<tr><td colspan="16">Es folgen noch 63 weitere Parametereinträge</td></tr>
</table>

Tabelle 3.3: Telekommando zum Setzen von Tabellen–Blöcken

Tabellen–Nummer	Blocknummer	Block von 128 Bytes
N	1	Block 1
	2	Block 2
	3	Block 3
	...	
	n	Block n

Tabelle 3.4: Struktur einer Parametertabelle

Validierung der Parameter Jeder Parameter muss von der Bodenstation überprüfbar sein. Um dies zu gewährleisten, sind für die eben beschriebenen Parameter folgende Ansätze gemacht worden. Die Flags erscheinen in der periodisch ausgelesenen Housekeeping–Telemetrie. Daher ist kein besonderes Telekommando notwendig, um den momentanen Wert dieser Parameter zu erfahren. Die Schwellwerte hingegen werden nur gelegentlich geändert. Hier genügt es, wenn zu jedem Schwellwert ein Telekommando existiert, um ihn auszulesen. Diese Telekommandos sind alle vom Typ »Lese Task Parameter« TC(5,4), dessen TID/FID dieselbe wie die zum Setzen des Parameters ist. Auf dieses

Telekommando hin wird ein Telemetriepaket TM(5,4) abgeschickt, welches den Wert des Parameters enthält.

Bei Tabellen wird folgendermaßen vorgegangen:

1. Beim Setzen eines Tabellenblockes mit einem »Lade Task Paramter«–Telekommando wird ein Telemetriepaket TM(5,4) erzeugt, das den neuen Inhalt des Tabellenblocks enthält.

2. Durch das Telekommando »Lese Task Parameter« mit der TID/FID einer Tabelle wird die ganze Tabelle ausgelesen und in Telemetriepaketen vom Typ TM(5,4) abgeschickt. Es sind die gleichen Telemetriepakete wie die, die beim Setzen von Blöcken erzeugt werden, aber es können bis zu drei Tabellenblöcke in einem solchen Paket auftauchen.

Es wäre am günstigsten gewesen, wenn jeder Tabellenblock mit einem Telekommando vom Typ »Lese Task Parameter« ausgelesen werden könnte. Leider kann man aber einem solchen Telekommando keinen Parameter anhängen, der etwa die Blocknummer enthält, die ausgelesen werden soll.[5]

Nachteilig an dieser Tabellenverwaltung ist, dass keine einzelnen Tabellenblöcke überprüft werden können, sondern immer die komplette Tabelle ausgelesen werden muss, was auch bei maximaler Telemetrierate bis zu etwa 2 Minuten dauern kann.[6] Es ist jedoch eine Implementierung eines Telekommandos geplant, das einen einzelnen Block mit einem Telekommando vom Typ »Lade Task Parameter« ausliest. Möglicherweise kann diese Funktion auch durch ein im folgenden beschriebenes Dump–Kommando durchgeführt werden.

Kontext–Tabellen Wenn ein Subinstrument abgeschaltet worden ist, müssen nach dem Anschalten alle Betriebsparameter, wie etwa eine Liste deaktivierter Pixel, wieder neu geladen werden. Dies könnte von der Bodenstation aus geschehen, würde aber einen großen Aufwand bedeuten. Einfacher ist es, wenn die DPE, die nicht abgeschalten wird, diese Parameter zwischenspeichert und nach dem Anschalten des Subinstrumentes zurückschreibt. Diese Parameter werden **Instrument–Kontext** genannt und sind aus Sicht der IASW eine Sonderform von Tabellen. Sie werden genau wie die Parametertabellen verwendet, haben aber einige zusätzliche Eigenschaften:

- Zusätzlich zur Blocknummer gibt es für jeden Block eine **Blocklänge**, die bestimmt, wieviel von dem Tabellenblock überhaupt verwendet wird. Dadurch können Tabellen beliebiger Größe mit bis zu 255 Blöcken verwaltet werden.[7] Die Blocklänge hat einen Standardwert, der aber verändert werden kann.

[5] Es wäre möglich, die Blöcke mit (TID/FID)–Paaren zu adressieren. Es wären damit $2^{16} = 65536$ Blöcke unterscheidbar. Allerdings müssten Parametertabellen mit mehr als 256 Blöcken auf verschiedene Task–Identifier aufgeteilt werden, was nachteilhaft für die Tabellenstruktur wäre. Schliesslich sollte den Task–Identifier eine bestimmte Bedeutung verliehen werden, die bei einer solchen totalen Belegung verloren gehen würde.

[6] Dies errechnet sich aus der größten Tabelle von 340kWord, 184 Telemetriepaketen pro 8 Sekunden maximal.

[7] Parametertabellen haben immer eine Größe von Vielfachen von 128 Byte und benötigen daher keine solche Größenanpassung.

- Für jede Kontexttabelle kann eine Prüfsumme ermittelt werden. Diese ist notwendig, um das erfolgreiche Sichern und Zurückschreiben des Kontextes in einem Subinstrument zu verifizieren. Diese Prüfsumme kann mit einem »Lese Task Parameter«–Telekommando von der Bodenstation ausgelesen werden.

- Der Inhalt der Kontexttabellen wird von der IASW als eine Folge von Words ohne besondere Struktur behandelt. Es liegt in der Verantwortung der Bodenstation, vernünftige Parameter zu setzen, da die IASW keine Konsistenzprüfung, außer mit eben beschriebener Prüfsumme, durchführt.

Die Struktur des Telekommandos zum Setzen von Kontext–Blöcken ist in Tabelle 3.5 wiedergegeben. Einzelheiten sind in [33] beschrieben.

64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
TID								FID							

72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87
Blocklänge								Blocknummer							

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
1.Parametereintrag															
Es folgen noch 63 weitere Parametereinträge															

Tabelle 3.5: Telekommando zum Setzen von Kontext-Tabellen–Blöcken

3.1.2 Broadcast–Pakete

Unter einem **Broadcast–Paket** (**BCPK**) versteht man eine Information, die auf dem Satelliten von der CDMU an alle Instrumente geschickt wird. Ein Teil davon wird auf INTEGRAL gemessen (Strahlung, Notabschaltung), ein Teil betrifft Daten des Orbits (Ekliptik, Strahlungsgürtel) und ein Teil wird von der Bodenstation gesetzt (etwa Dauer des Pointing).

Auf INTEGRAL hat dieses Paket die Form eines normalen Telekommandos vom Typ TC (15,1), das alle acht Sekunden an alle Instrumente abgeschickt wird. Die IASW kann daraus die Information des aktuellen Satellitenstatus extrahieren. Die Struktur ist (für IBIS leicht gekürzt) in Tabelle 3.6 dargestellt, genauere Informationen sind in [22] und [2] zu finden.

3.1.3 Automatismen

Einige Steuerungsaufgaben können nur schlecht von der Bodenstation aus bewältigt werden. Dazu gehört zum einen die Konfigurierung einer Messung, wenn sich der Satellitenzustand (Point/Slew etc.) ändert. Zum anderen gibt es aber auch unerwartete Ereignisse, auf die die IASW mit geeigneten Maßnahmen

Name	Position (Word)	Bit (Start)	Länge (Bits)	Beschreibung
B1	1	0	32	Eintrittszeit in den Strahlungsgürtel
B2	3	0	32	Austrittszeit aus dem Strahlungsgürtel
B3	5	0	32	Startzeit der Eklipse
B5	7	0	32	Endzeit der Eklipse
B4	9	0	1	Strahlungsmessung gültig–Flag
C8	14	0	32	Bezeichner des aktuellen Pointing–Status
C9	16	0	32	Dauer des aktuellen Pointing
C2	18	8	8	Verfügbarer Telemetrie–Anteil für IBIS
C6	20	8	1	Ausschalten der Instrumente (von der Bodenstation gesetzt)
C7	20	9	1	Wechsel der Empfangsstation bzw. Hand Over
A5	27	0	16	Strahlungsrate 1
A6	28	0	16	Strahlungsrate 2
A7	29	0	16	Strahlungsrate 3
A1	30	0	1	Instrument ausgerichtet (On Target)
A3	30	12	4	ACC-Modus (Orbit–Satellitenzustand)
A2	31	2	1	Notausschaltung–alle Instrumente sichern.
A4	31	2	1	Point–Slew–Status

Tabelle 3.6: Struktur eines Broadcast–Pakets

reagieren muss. Viele solcher Ereignisse werden über Broadcast–Pakete mitgeteilt (siehe Tabelle 3.6). Die Software muss auf folgende Ereignisse reagieren können:

- **Eklipse** — Der Satellit wird von der Erde abgeschattet, und sein Stromverbrauch muss reduziert werden. Dabei werden alle Subinstrumente, nicht aber die DPE abgeschalten. Vor dem Abschalten muss die IASW den Kontext aller Subinstrumente sichern, nach dem Anschalten muss er wiederhergestellt werden und die Subinstrumente in einen Standard–Modus gebracht werden.

- **Strahlungsgürtel** (Radiation Belt) — Wenn sich der Satellit in dem Strahlungsgürtel der Erde befindet, muss eine laufende Messung beendet werden, und die Detektoren müssen sich vor der Strahlung schützen. Letzteres heißt, dass von dem VETO–Instrument die Hochspannung der Photomultiplier abgeschalten werden muss.

- **Erhöhte Strahlung** (Verstärkte Sonnenaktivität) — Auch wenn die Strahlung zu stark ansteigt muss die Messung beendet und die Hochspannung von VETO abgeschalten werden. Die Strahlung wird auf INTEGRAL gemessen und in einem Broadcast–Paket mitgeteilt. Die Schwelle, ab der eine Strahlung als erhöht gewertet wird, läßt sich in einer Parametertabelle einstellen.

- **Notausschaltung** Wenn das Broadcast–Paket eine Notausschaltung (Emergency Save Acquisition) oder eine sofortige Beendigung (Imminent Switch off) mitteilt, muss die Software die Messsung beenden, die VETO–Hochspannung abschalten und im ersteren Fall den Kontext sichern.

- **Resynchronisation von der HEPI** — Wenn die Daten, die die IASW von der HEPI einliest, ein falsches Format haben, oder wenn die Datenrate unter einer bestimmten Schwelle liegt, initialisiert die IASW die HEPI neu und synchronisiert sie mit der OBT.

- **Noisy Pixel** Bei PICsIT und ISGRI können einzelne Pixel–Elemente ausfallen und eine permanente, hohe Datenrate erzeugen. Dies würde die Messung erheblich stören, da dann die wissenschaftlichen Daten zurückgedrängt werden. In ISGRI ist ein Mechanismus zur Behandlung solcher rauschenden Pixel (Noisy Pixel) bereits implementiert. Bei PICsIT dagegen muss die IASW auf die Datenerzeugung achten und gegebenfalls einzelne Pixel auf dem PICsIT–Detektor ausschalten. Das genaue Vorgehen ist in [36] beschrieben.

- **Point–Slew** — Wenn der Satellit seinen Status aufgrund des Ditherns zwischen Slew–Mode und Pointing–Mode wechselt, muss die Konfiguration einer Messung geändert werden. Es macht z.B. keinen Sinn, Histogramme im Slew–Mode zu erstellen.

Alle diese Steuerungsaufgaben der IASW heissen **Automatismen**, da sie auch bei totalem Ausfall der Verbindung zur Bodenstation aktiv sein können. Sie müssen daher unkritisch sein, d.h. die Instrumente dürfen bei einer Fehlfunktion nicht beschädigt werden. Einige von ihnen müssen die Hardware überwachen und müssen daher zuverlässig reagieren. Allen Automatismen ist gemeinsam, dass sie abschaltbar sind, um bei einer Fehlfunktion keinen Datenverlust zu erleiden. Eine Sonderstellung hat der Point–Slew–Automatismus, weil er die Messfunktionen umkonfiguriert statt eine Messung generell zu beenden. Daher sollte seine Funktion, auch wenn er abgeschaltet ist, durch explizit von der Bodenstation geschickte Telekommandos ersetzt werden können. Dies geschieht mit einem Telekommando »Modus ändern«, dessen letzter Parameter bestimmt, ob die IASW sich im Pointing Mode (1) oder Slew Mode (2) befindet. Wenn ein Automatismus aktiv wird, wird dies durch noch zu besprechende Ereignismeldungen der Bodenstation mitgeteilt.

3.2 Housekeeping

Für eine aktive Steuerung ist nach den Gesetzen der Kybernetik eine Rückkopplung notwendig. Das heißt für den Fall eines Satelliten, dass die Bodenstation über den Status im Satelliten Bescheid wissen muss, wenn sie das Instrument erfolgreich über Telekommandos bedienen will. Deshalb hat sich in der Satellitentechnik der Terminus des **Housekeeping** eingebürgert, der (wie bei einer guten Hauswirtschaft) regelmäßig Rechenschaft von dem Zustand des Satelliten abgibt. Konkret heißt das für IBIS, dass alle acht Sekunden ein besonderes mit Statuswerten gefülltes Telemetriepaket abgeschickt wird. Es wird als **HK**–Paket oder **Housekeeping–Paket** bezeichnen. Es enthält in verschiedene Blöcke aufgeteilt Informationen über die CSSW, über jedes der Subinstrumente und über die IASW.

Während die CSSW ihren Anteil des HK–Pakets selber beschreibt, muss die IASW sowohl ihre eigenen Statuswerte zusammenfassen, wie auch die Housekeeping–Daten von den Subinstrumenten auslesen. Dies geschieht alle acht Sekunden. Damit die Housekeeping–Daten möglichst aktuell sind, ist das Auslesen mit dem Abschicken so synchronisiert, dass das Einlesen etwa vier Sekunden vor dem Abschicken beginnt.

Ereignismeldungen Für wichtige, nichtperiodische Ereignisse gibt es in dem Housekeeping–Paket Felder, die mit **Ereignismeldungen** (On–Event–Messages — **OEM**) gefüllt werden können. Der Aufbau einer solchen Er-

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td colspan="16">MSW der An–Bord–Zeit</td></tr>
<tr><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td colspan="16">LSW der An–Bord–Zeit</td></tr>
<tr><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td><td>45</td><td>46</td><td>47</td></tr>
<tr><td colspan="6">Message 1</td><td colspan="2">Class</td><td colspan="8">Identifier</td></tr>
<tr><td>48</td><td>49</td><td>50</td><td>51</td><td>52</td><td>53</td><td>54</td><td>55</td><td>56</td><td>57</td><td>58</td><td>59</td><td>60</td><td>61</td><td>62</td><td>63</td></tr>
<tr><td colspan="16">Message 2</td></tr>
<tr><td>64</td><td>65</td><td>66</td><td>67</td><td>68</td><td>69</td><td>70</td><td>71</td><td>72</td><td>73</td><td>74</td><td>75</td><td>76</td><td>77</td><td>78</td><td>79</td></tr>
<tr><td colspan="16">Message 3</td></tr>
</table>

Tabelle 3.7: Struktur einer Ereignismeldung

eignismeldung ist in Tabelle 3.7 wiedergegeben. Sie unterscheiden sich durch eine Einteilung in Klassen (**OEM–Class**) und Bezeichner (**OEM–Identifier**). Drei Felder (Message 1 - Message 3) enthalten spezifische Information; die ersten zwei Word enthalten die Zeit des Auftretens im OBT–Format[8]. Die CSSW, die IASW und jedes Subinstrument darf beliebig viele solche Ereignismeldungen erzeugen. Pro Housekeeping–Zyklus werden dabei bis zu acht Ereignismeldungen von den Subinstrumenten eingesammelt. Da bloß acht Ereignismeldungen in ein Housekeeping–Paket passen, werden sie in der CSSW in einem Ringpuffer (FIFO) mit einer Kapazität von 64 Ereignismeldungen zwischengespeichert. Die meisten Ereignismeldungen sind für Fehlermeldungen vorgesehen, es gibt aber auch Warnungen und Statusmeldungen.

Zusätzliche Housekeeping Für größere Mengen an Statusberichten werden zusätzliche Housekeeping–Pakete (**Additional Housekeeping**) abgeschickt, die in Intervallen von 30 Sekunden bis 30 Minuten von ISGRI, PICsIT und VETO gelesen werden. Außerdem schickt die IASW ein zusätzliches HK–Paket (auch **Science HK** genannt), wenn folgende Ereignisse auftreten:

- Start des wissenschaftlichen Modus
- Ende des wissenschaftlichen Modus
- Start einer Histogramm–Integration
- Ende einer Histogramm–Integration
- Start des Pointing
- Start des Slew
- Verlust von Daten infolge Telemetrieüberlauf
- Auf Anfrage via Telekommando
- Synchronisation von HEPI

In diesem Telemetriepaket sind wesentliche Informationen über die wissenschaftlichen Messdaten enthalten.

3.3 Verarbeitung der Einzel–Ereignisse

IBIS kennt zwei verschieden Arten, wie Messergebnisse gewonnen werden können. Die eine ist, dass jedes **Ereignis** (also die Meldung eines Detektorpixels, dass ein Gamma-Quant oder geladenes Teilchen interagiert hat) mit allen relevanten Parametern (Zeit, Energie, Ort) aufgezeichnet und zur Bodenstation geschickt wird. Diese Einzelereignisse werden auch als Events bezeichnet.

Die andere Möglichkeit ist die, dass alle Einzelereignisse in Histogrammen gesammelt werden. Natürlich sind Einzelereignisse vorzuziehen, da bei ihnen keine Information durch eine Integration verloren geht. Da aber die Bandbreite der Verbindung Satellit–Erde auf maximal 85 kbps begrenzt ist, wäre schon bei

[8] Das letzte Word der OBT wird ignoriert.

einer Rate von 1250 Photonenereignissen pro Sekunde[9] die maximal verfügbare Kapazität überschritten.

Bearbeitungsschritte der Einzelereignisse Alle Einzelereignisse kommen von der HEPI. Wenn die HEPI mehr als 1000 Word an Einzelereignissen bearbeitet und im FIFO abgelegt hat, können diese von der DPE ausgelesen werden. Die IASW muss diese dann nach dem Typ des Ereignisses trennen, möglichst gut komprimiert in ein Telemetriepaket schreiben und dieses abschicken. Jeder Typ eines Einzelereignisses ist einem bestimmten Typ/Subtyp des Telemetriepaketes zugeordnet.

Wenn wegen erhöhtem Datenaufkommen gerade keine Telemetriepakete eines bestimmten Typs abgeschickt werden können, muss die Bearbeitung (nicht aber das Einlesen!) der zugehörigen Einzelereignisse unterbrochen und die Einzelereignisse selber verworfen werden. Dies ist notwending, damit auf jeden Fall die Ereignisse von HEPI zur DPE gelangen und es keinen Überlauf der Detektorpuffer, aus denen die HEPI ihre Daten einliest, gibt. Der FIFO–Puffer der HEPI kann nicht überlaufen, da diese die Datenbearbeitung vorher anhält.

Einzelereignisse In der HEPI und IASW werden folgende Einzelereignisse verarbeitet:

- Einzelereignisse, die von ISGRI kommen (CdTe–Single–Events).
- Einzelereignisse, die von PICsIT kommen (CsI–Single–Events).
- PICsIT–Mehrfachereignisse, bei denen HEPI die Energie korrigiert. (CsI–Multiple–Events)
- Kalibrationsereignisse, die von ISGRI kommen (CdTe–Calibration–Events).
- Ereignisse, die transparent von ISGRI und PICsIT durch die HEPI laufen. (CdTe–Raw–Events bzw. CsI–Raw–Events).
- Compton–Ereignisse, die zeitgleich in ISGRI und PICsIT aufgetreten sind (CdTe+CsI–Single–Events bzw. CdTe+CsI–Multiple–Events). Es wird für PICsIT zwischen Einzel– und Mehrfachereignissen unterschieden. Der Name rührt daher, dass Photonen unter Energieabgabe an der einen Detektorebene comptongestreut werden können, und dann von der anderen Ebene absorbiert werden. Aus dem Energiedeposit und der Position lassen sich dann Rückschlüsse über die Richtung und Energie des Photons bestimmen.
- Ein Histogrammtyp, der die Ortsinformation verwirft und nur auf die Energie achtet (**Spektral Timing–Events**). Dabei sammelt die HEPI über eine Zeit von 1 msec bis 500 msec Einzelereignisse in acht verschiedenen Energiekanälen. Die Komprimierung der acht Zellen geschieht durch

[9] Das errechnet sich aus der maximalen Telemetrierate von 23 Paketen pro Sekunde, 217 Word pro Paket und 4 Word pro Einzelereignis = 1242 cts.

Reduktion jeder Zelle auf ein 4-Bit-Feld, wenn ihr Inhalt kleiner als 15 ist.

Bis auf den letzten Typ werden die Einzelereignisse verlustlos mit einer Differenzmethode für die Darstellung des Zeit–Feldes komprimiert und in die entsprechenden Telemetriepakete gepackt. Die Formate und Algorithmen sind in [27] ausführlich beschrieben. Die Bedeutung der Einzelereignisse findet sich in [35, 34].

3.4 Verarbeiten von Histogrammen

3.4.1 Histogrammtypen

In den Histogrammen, die als Speicherbereiche in der HEPI angelegt sind, können Ereignisse aufaddiert werden. Dies wird als **Integration von Histogrammen** bezeichnet. Die Ereignisse werden nach Pixel–Adresse (Ort) und Energie in verschiedene Kanäle sortiert. Histogramme werden nur für Ereignisse von PICsIT verwendet, da für ISGRI eine wesentlich geringere Ereignisrate erwartet wird. Es gibt folgende Typen von Histogrammen:

- **CsI–Einzelereignis–Histogramm**: Summiert alle Einzelereignisse von PICsIT auf, die keine Kalibrationsereignisse sind.
- **CsI–Mehrfachereignis–Histogramm**: Alle Mehrfachereignisse von PICsIT werden hier aufintegriert.
- **CsI–Polarimetrie–Histogramm**: Alle PICsIT–Mehrfachereignisse, die benachbart sind und damit eine Polarisationsrichtung aufweisen (können), kommen in dieses Histogramm. Dieses Histogramm hat für acht verschiede Polarisationsrichtungen eine eigene Histogrammzelle.
- **Kalibrations–Histogramm**: Alle Kalibratonsereignisse von PICsIT gelangen in dieses Histogramm.

Die Anordnung des Histogramm-Speichers in der HEPI ist in Tabelle 3.8 festgehalten.

3.4.2 Erzeugung von Histogrammen

Anders als bei den Einzelereignissen muss bei Histogrammen ein umfangreiches Protokoll berücksichtigt werden, damit ein komplettes Bild integriert und zuverlässig zur Bodenstation geschickt werden kann. Die Schritte sind folgende:

1. **Histogramm Löschen** — vor der Integration eines Histogramms muss die vorhergehende Messung gelöscht werden.
2. **Histogramm–Integration** — die HEPI muss so konfiguriert sein, dass sie das gewünschte Histogramm integriert. Dies geschieht mit einem Befehl über die serielle Schnittstelle, welcher das HEPI–Statusregister ändert.

Adresse	CsI Ereigniss aktiv	Polarimetrie aktiv
0 MByte	CsI-Histogramm Für Einzelereignisse Puffer 1 1 MByte	Polarimetrie–Histogram Puffer 1 2 MByte
1 MByte	CsI-Histogramm Für Mehrfachereignisse Puffer 1 1 MByte	
2 MByte	CsI-Histogramm Für Einzelereignisse Puffer 2 1 MByte	Polarimetrie–Histogram Puffer 2 2 MByte
3 MByte	CsI-Histogramm Für Mehrfachereignisse Puffer 2 1 MByte	
4 MByte	CsI-Histogramm Für Kalibrationsereignisse Puffer 1 0.25 MByte	CsI-Histogramm Für Kalibrationsereignisse Puffer 1 0.25 MByte
4.25 MByte	CsI-Histogramm Für Kalibrationsereignisse Puffer 2 0.25 MByte	CsI-Histogramm Für Kalibrationsereignisse Puffer 2 0.25 MByte
4.5 MByte		

Tabelle 3.8: Histogramm–Speicheranordnung in HEPI

3. Ist die Integrationszeit vorüber, so wird die Histogramm–Integration wieder gestoppt.

4. **Histogramm Auslesen** — Nach der Integration ist das Histogramm auslesebereit. Dazu wird ein Befehl über die serielle Schnittstelle an die HEPI geschickt, woraufhin sie einen **Histogramm–Block** von 1 kByte im Ausgabe–FIFO zur Verfügung stellt. Die IASW kann dann den Block über die serielle Schnittstelle auslesen.

5. **Histogramm–Komprimierung** — Wenn ein Histogramm–Block zur IASW gelangt ist, muss sie diesen möglichst gut komprimieren.

6. Schliesslich kann der komprimierte Histogrammblock an die Bodenstation geschickt werden. Dabei dürfen anders als bei Einzelereignissen keine Daten infolge hoher Telemetrierate verlorengehen.

Bei diesem Ablauf gibt es mehrere Randbedingungen. Zunächst darf keine Integration von Histogrammen erfolgen, wenn der Satellit im Slew–Mode ist. Denn andernfalls würde die Richtungsinformation verlorengehen. Dies gilt nicht für Kalibrations–Histogramme, da die Kalibrationsquelle ortsfest auf dem Satelliten montiert ist und sich mitbewegt. Es wird daher zwischen **Science–Histogrammen** (CsI–Einzel/Mehrereignis–Histogrammen und Polarimetrie–Histogrammen) und **Kalibrations–Histogrammen** unterscheiden. Eine Integration von Science–Histogrammen wird unterbrochen, wenn der Satellit über BCPK bekanntgibt, dass die Ausrichtung nicht stabil ist.

In der HEPI ist der Histogrammspeicher doppelt ausgelegt, um während einer Integration das vorherige Histogramm auslesen zu können. Dazu bedarf es einer zuverlässigen Speicherverwaltung. Es können entweder CsI–Ereignisse oder Polarimetrie–Histogramme gebildet werden, da diese Histogramme den gleichen Speicherbereich belegen. Es können also nicht alle Histogrammfunktionen gleichzeitig benutzt werden. Die Speicherverwaltung muss außerdem berücksichtigen, dass die Integrationszeit von Science–Histogrammen und Kalibrationshistogrammen verschieden sein kann, so dass die Integration und der Auslesevorgang dieser beiden Arten von Histogrammen entkoppelt sein muss.[10]

Histogramme könnten mit einer relativ hohen Geschwindigkeit ausgelesen und verarbeitet werden.[11] Für eine möglichst konstante Auslastung der Telemetrie sollte das Auslesen gleichmäßig über den Integrationszeitraum des nächsten Histogramms erfolgen. Daher muss die Ausleseroutine wissen, wie lange die aktuelle Integration wahrscheinlich dauern wird.

3.4.3 Vorgaben für Histogrammverarbeitung

Für die Histogrammverarbeitung werden folgende Informationen benötigt:

- **Point–Slew–Status**: wird aus dem Broadcast–Paket gelesen, sofern der Point–Slew-Automatismus an ist. Andernfalls kann er via Telekommando TC(5,5) geändert werden.
- **On Target**: Ein Flag des Broadcast–Pakets, das die stabile Ausrichtung des Satelliten signalisiert.
- **Eingestellte Integrationszeit**: Kann als Parameter mit einem Telekommando für Science– und Kalibrationshistogramme konfiguriert werden.
- **Slew–Zeit**: Bestimmt anhand des Wechsels vom Slew zum Pointing und der Pointing–Dauer, wann der nächste Slew–Mode des Satelliten beginnt. Sie ist undefiniert, wenn der Point–Slew–Automatismus ausgeschalten ist.
- **Hand–Over**: Ein Flag des Broadcast–Pakets, das ein Umschalten der Bodenstation mitteilt. Ist es gesetzt, können unter Umständen Telemetriepakete verloren gehen.

Mit diesen Informationen versucht der Histogramm–Verarbeitungsalgorithmus möglichst ohne Verlust an Beobachtungszeit Histogramme mit der eingestellten

[10] Zu beachten ist dabei, dass die Integrationszeit der Science–Histogramme gemeinsam die gleiche ist, da ja nur entweder Polarimetrie oder CsI–Ereignisse integriert werden können.

[11] Gemäß dem aktuellen Design könnte jedes RTC (125 msec) ein Histogrammblock von 1024 Byte gelesen werden, bei optimaler Telemetrierate ist dann ein Science–Histogramm nach 256 Sekunden (4 Minuten) ausgelesen und verarbeitet.

Integrationszeit zu erzeugen und gleichmäßig auszulesen. Dies erfordert folgende Bedingungen:

Integration von Science–Histogrammen Die Histogramme für CsI-Ereignisse und für die Polarisationsereignisse schließen sich gegenseitig aus. Dennoch ist die Handhabung bezüglich Start, Dauer und Ende der Integration für beide gleich:

- **Start** Die Integration von Science–Histogrammen beginnt, wenn:
  - Der Satellit im Pointing–Mode ist.
  - Der Satellit On Target ist.
  - Es keine Science-Histogramme integriert werden.
  - Ein Puffer (1 oder 2) leer (vollständig ausgelesen) ist.
  - Bei eingeschaltenem Point–Slew–Automatismus ein Histogramm mit der eingestellten Integrationszeit im aktuellen Pointing–Mode integriert werden kann.
- Die **Integrationsdauer** bestimmt sich aus der eingestellten Integrationszeit und der verbleibenden Zeit im Pointing–Mode gemäß:
  - Wenn der Point–Slew–Automatismus nicht eingeschalten ist, so ist die Integrationsdauer die der eingestellten Integrationszeit.
  - Wenn der Point–Slew–Automatismus an ist und im aktuellen Pointing–Mode zwei Histogramme mit der eingestellten Integrationszeit integriert werden können, so wird diese als Integrationszeit verwendet.
  - Andernfalls wird bis zum Slew–Beginn integriert.
- Die Integration erfährt eine **Unterbrechung**, wenn der Satellit nicht On Target ist. Wenn der Satellit wieder stabilisiert ist, muss die Integration fortgeführt werden (sofern sie nicht generell gestoppt ist) und die Unterbrechungszeit bestimmt werden. Unterbrechungen sind unabhängig vom Point–Slew–Automatismus.
- Das **Ende** der Integration ist erreicht, wenn:
  - Die Integrationszeit abgelaufen ist, oder
  - Der Slew–Mode angefangen hat.[12]
  - Der Science–Mode der IASW verlassen wurde.
- Die **Tatsächliche Integrationszeit** bestimmt sich aus der Differenz zwischen Ende und Beginn einer Integration abzüglich der Zeit, die die Integration unterbrochen war.

[12] Auch der Beginn des Slew–Mode kann bei abgeschaltetem Point–Slew–Automatismus via Telekommando gesetzt werden.

Integration von Kalibrations–Histogrammen Bei den Kalibrations–Histogrammen muss nicht auf den Pointing–Mode des Satelliten geachtet werden. Dies vereinfacht die Behandlung beträchtlich:

- Kalibrations–Histogramme **starten**, wenn ein Puffer für Kalibrationshistogramme (1 oder 2) frei ist und nicht gerade eine Integration von Kalibrationshistogrammen läuft.
- Die **Integrationsdauer** richtet sich nur nach der eingestellten Integrationszeit.
- Das **Ende** der Integration ist erreicht, wenn
  - Die Integrationszeit vorüber ist.
  - Der Science–Mode der IASW verlassen wurde.

Auslesen von Histogrammen An das Auslesen sowohl von Science–Histogrammen wie auch Kalibrationshistogrammen werden Anforderungen gestellt, die hier aufgelistet sind:

- Science–Histogramme und Kalibrationshistogramme werden getrennt ausgelesen, da sie verschiedene Integrationszeiten haben können.
- Das Auslesen eines Histogramms beginnt immer nur beim Start einer Integration eines neuen Histogramms gleichen Typs.
- Histogramme sollen mit einer im Mittel konstanten Datenrate über den Zeitraum der gerade laufenden Integration ausgelesen werden, so dass die Belastung der Telemetrie möglichst gut verteilt ist. Diese Datenrate wird beim Start des Auslesens eines neuen Histogramms anhand der Integrationszeit des nächsten Histogramms und der Größe des auszulesenden Histogramms berechnet.
- Das Auslesen von Science–Histogrammen soll nur im Pointing–Mode geschehen. Wenn der Satellit in den Slew–Mode geht, muss das Auslesen von Science–Histogrammen bis zum Beginn des nächsten Pointing–Mode unterbrochen werden.
- Wenn die IASW den Science–Mode verlasssen hat, muss das Auslesen unterbrochen werden. Wenn sie wieder in ihn wechselt, muss das Auslesen fortgesetzt werden.[13]
- Es dürfen keine Histogrammblöcke verlorengehen. Wenn daher die Telemetrie überlastet ist bzw. der TM–Puffer voll ist, muss das Auslesen angehalten werden.[14]

[13]Der Nachteil dabei ist, dass die Histogramme unter Umständen nicht mehr korrekt sind, weil der Speicher in HEPI nicht besonders (mit einer Error–Correction) gegen Strahlung geschützt ist.

[14]Um eine totale Blockierung zu vermeiden, ist eine Wartezeit von 10 RTC vorgesehen. Wenn diese überschritten ist, wird das Telemetriepaket mit einer Ereignismeldung verworfen.

- Während des Hand Over dürfen keine Histogramm–Telemetriepakete gesendet werden.
- Das Auslesen von Histogrammen ist beeendet, wenn der entsprechende Puffer leer ist.
- Das Auslesen von Histogrammen kann im Standby–Mode der IASW mit einem Telekommando provoziert werden. Die Lesegeschwindigkeit entspricht der beim Start vorausberechneten Integrationszeit.

3.5 Software Patchen und Dumpen

Eine Software, die einerseits wichtige Funktionen zu übernehmen hat, und deren Zugriff anderseits auf Telekommandos beschränkt ist, sollte in der Lage sein, eine beliebige Änderung auch während der Mission zuzulassen. Man nennt dies **Patchen** von Software.[15]

Patchen der Subinstrumente Für die IASW muss das Problem aufgeteilt werden. Denn das IBIS–Instrument enthält mehrere Computer, deren Software änderbar sein muss. Die Subinstrumente sind für die Möglichkeit zum patchen selbst verantwortlich; die IASW stellt ihnen die entsprechenden Kommandos zu. Das betrifft allerdings nur PICsIT und VETO, da HEPI keine Software enthält und ISGRI die Funktionalität so eingeschränkt hat, dass hier kein Patchen nötig oder möglich ist.

Patchen der IASW Die IASW selber hat keine Möglichkeit beliebige Speicherzellen der DPE zu ändern. Dies ist nicht notwendig[16], da die CDMU über den OBDH–Bus in der Lage ist, jede DPE von INTEGRAL zu patchen. Dies ist ein Nebenprodukt von der Tatsache, dass Telekommandos direkt von außen in den Speicher einer DPE geschrieben werden. Diese Funktionalität ist dahingehend erweitert, dass jeder Wert an eine beliebige Adresse einer DPE gesetzt werden kann. Die einzige Vorraussetzung der DPE ist, dass sie dabei angehalten wird.

Dumpen Für eine Kontrolle sollten alle Speicherbereiche auslesbar sein. Dieser Vorgang wird **dumpen**[17] genannt. Dies ist weit unproblematischer als das patchen, weil keine Funktionalität geändert wird. Für das IBIS–Instrument gibt es für einen Dump mehrere Möglichkeiten:

---

[15] Der Begriff kommt eigentlich aus der Hackerscene und bezeichnet das Ändern von Programmcode auf Maschinensprachebene mit Methoden, die wenig mit dem Vorgehen des Software engineering gemeinsam haben.

[16] Es gab Überlegungen einer Implementierung der »indirekten Sprünge«, bei der eine Funktion durch eine andere ersetzt werden kann, indem ihre Einsprungadresse geändert wird. Genauere Betrachtungen lassen aber fragen, ob dies tatsächlich sinnvoll ist. Denn im Normalfall ist die IASW mit Hilfe des sehr flexiblen Tabellenkonzeptes in weiten Grenzen konfigurierbar. Wenn aber grundsätzliche Änderungen vorgesehen sind, ist es besser, sie auf dem Boden zu testen und dann via CDMU die notwendigen Änderungen in die An–Bord–IASW zu schreiben.

[17] Aus dem Englischen bezeichnet *dumpen* verschleudern oder auch Schutt abladen [12].

- Alle Subinstrumente (bis auf HEPI) können Speicherauszüge mittels Telekommandos generieren.
- Die IASW unterstützt das Auslesen aller Parameter, was einem Dumpen auf höherer Ebene gleichkommt.
- Die CSSW hat eine Funktion, mit der beliebige Speicherbereiche der DPE auslesbar sind.
- Via CDMU kann — analog zum Patchen — ein beliebiger Speicher ausgelesen werden.[18]

3.6 Schutz vor Fehlverhalten

Bis hier wurde beschrieben, welche Anforderungen an die IASW normalerweise gestellt sind. Bei Betrachtungen eines **Fehlverhaltens** müssen zwei Arten von Fehlern auseinandergehalten werden. Denn es gibt Fehler, die wohldefiniert sind und eine besondere Reaktion der Software erfordern. Dazu gehört etwa der Umgang beim Versagen der Hardware oder die Nichtverfügbarkeit von Resourcen.

In diesen Fällen darf die Software nicht in einen undefinierten Zustand gehen, sondern muss mittels einer geeigneten Strategie versuchen, auf diese Situation zu reagieren. In den meisten Fällen genügt ein Fehlerreport mit einer Ereignismeldung und ein Abbrechen der gerade vorgenommenen Operation. Dies ist ein **definierter Umgang mit Fehlverhalten**.

Auf der anderen Seite gibt es Softwarefehler, die als »Bugs« bezeichnet werden und sich entweder auf eine mangelhafte Spezifikation oder auch eine nicht der Spezifikation angepasste Implementierung zurückführen lassen. Wenn beispielsweise die Bestimmung einer Prüfsumme nicht korrekt definiert worden ist, so schlägt der Umgang mit Kontexttabellen aufgrund einer falschen Spezifikation fehl. Wenn dagegen etwa die Größe der Kontexttabelle — etwa wegen eines Tippfelhers — nicht korrekt programmiert ist, dann gibt es auch hier ein Fehlverhalten; diesmal allerdings wegen einer nicht konformen Umsetzung der Spezifikation.[19]

Um diese Sorte von Fehlern, die **intrinsische Fehler** genannt werden, zu vermeiden, werden veschiedene Strategien bei der Softwareentwicklung durchgeführt. Sie werden in Kapitel 7.5 behandelt. Aber dennoch sollte auch die Software die Existenz solcher Fehler berücksichtigen und Maßnahmen ergreifen, damit auch bei solchen Fehlern die Bodenstation während der INTEGRAL–Mission eine Chance hat, die Ursache zu erkennen. Dazu gehört die Implementierung von Sicherheitsabfragen, die z.B. eine Division durch Null abfangen, und periodische Kontrollen, um etwa einen Stapelspeicherüberlauf zu entdecken. Was in der IASW dafür unternommen wurde, ist in Kapitel 5.8 dargestellt.

[18] Der Nachteil dieser Methode ist, dass für größere Speicherbereiche viele Telekommandos notwendig sind, da pro Telekommando nur ein Telemetriepaket generiert wird. Die Funktion der CSSW dagegen kann mit *einem* Telekommando eine ganze Folge von Telemetriepaketen erzeugen.

[19] Diese Beispiele sind nicht aus der Luft gegriffen, sondern sind bei den Integrationstests des Engineering Models von IBIS (EM) so aufgetreten.

Der definierte Umgang mit Fehlverhalten Wie schon angesprochen muss bei jeder Aktion, die auf die Hardware zugreift, auch die Möglichkeit in Betracht gezogen werden, dass der Zugriff versagt. Eine andere Ursache von Fehlverhalten liegt an der beschränkten Kapazität der Telemetrie: die Software muss damit umgehen können, dass das Telemetriepaketaufkommen zu groß ist. Konkret muss die IASW folgende Situationen behandeln können:

- **Versagen des Puls–Kommandos**: Da Pulse nur über Telekommando angesprochen werden, wird das Telekommando mit einer Ereignismeldung abgebrochen.
- **Versagen der Analogen Schnittstelle**: Analogwerte werden in das Housekeeping–Telemetriepaket geschrieben. Bei Versagen wird eine Ereignismeldung erzeugt.
- **Versagen der Low speed Serial Line**: Die aktuelle Operation wird mit einer Ereignismeldung abgebrochen und die LSL wird zurückgesetzt. Wenn die Operation eine Kommandofolge betrifft (etwa Kontext sichern), muss mit dem nächsten unabhängigen Teil fortgefahren werden.
- **Timeout der Low speed Serial Line**: Bedeutet meistens, dass ein Subinstrument keine oder zu wenig Daten auf eine Anfrage schickt. Behandlung wie beim Versagen der Low speed Serial Line.
- **Versagen/Timeout der High Serial Line**: Eine Ereignismeldung wird abgeschickt, und die Operation wird nach 125 msec wiederholt (Nur HBR B).
- **Versagen einer Relais–Leitung**: Es gibt keine Möglichkeit, diesen Fehler zu entdecken, außer im Fehlverhalten der Software (Daten werden nicht eingelesen).
- **Versagen der Telekommandos**: Wenn Telekommandos verloren gehen, so erzeugt die IASW keine Fehlermeldung. Über das Housekeeping–Paket kann dieser Fehler festgestellt werden.[20]
- **Versagen der Telemetrie**: Wenn der Telemetriepuffer überläuft, wird das Telemetriepaket verworfen, und eine Ereignismeldung erzeugt.[21]
- **Versagen des Housekeepings**: Ein Versagen der Housekeeping–Pakete der IASW kann von der IASW nicht entdeckt werden. Da ein solcher Fehler aber fast immer ein grundlegendes Versagen eines Softwareteiles bedeutet, sollte aus dessen Fehlerreport in einer Ereignismeldung die Ursache rekonstruiert werden können.[22]

[20]Die IASW könnte dieses Versagen entdecken, wenn eine Wartezeit (Timeout) definiert wäre, innerhalb der ein Telekommando eintreffen muß. In der aktuellen Version ist der Timeout jedoch auf »Unendlich« gesetzt.

[21]Standard–Housekeeping–Pakete werden unabhängig von dem Telemetriepuffer der CSSW abgeschickt. Die Ereignismeldung kann daher trotz Pufferüberlauf die Bodenstation erreichen.

[22]Gegen ein Deadlock müssen allerdings geeignete Maßnahmen getroffen werden. Siehe Kapitel 4.2.1.

- **Versagen der OEM**: Bei zu großem Aufkommen von Ereignismeldungen werden die letzten verworfen. Für eine Erkennung ist in den Housekeeping ein Zähler mit Überlaufbit vorgesehen.

Diese Punkte sollen grob beschreiben, wie die IASW sich im Fehlerfalle verhält. Für manche Fälle, wie die der Histogrammbehandlung, musste das Verhalten etwas erweitert werden. Siehe dazu die Implementierungsbeschreibung in Kapitel 4 und Kapitel 5.8.

Kapitel 4

Die Schnittstellen der IASW

Die umfassende Funktionalität der IASW ist nur schwer mit einem monolithischen Programm zu verwirklichen[1], so dass es unabdingbar ist, das Programm in weitgehend unabhängige **Module** aufzuteilen. Die IASW besteht aus vielen derartigen Einzelmodulen, die grob in solche, die mit der Außenwelt kommunizieren, und solche, die interne Datenbearbeitung vornehmen, aufgeteilt werden kann. Diese Module sind im wesentlichen als ADA–Packages implementiert, die in einer Datei eine Schnittstelle und in einer weiteren die Implementation definieren. Die im folgenden verwendeten Modulnamen entsprechen daher den Dateinamen des Quelltextes der IASW.

Zunächst sollen erstere, die **Schnittstellenmodule**, betrachtet werden. Dies hat für die Softwareentwicklung den Vorteil, dass alle internen Module sich nicht um Fragen des Hardwarezugriffs kümmern müssen, sondern stets eine einheitliche Oberfläche vorfinden. Wenn dies ordentlich geschehen ist, kann die DPE und das in ihr enthaltene Programm als »schwarzer Kasten«[2] agieren, der für gewisse Eingaben bestimmte Ausgaben produzieren soll.

Treten Hardwarefehler auf, so lassen sie sich schnell auf das entsprechende Ein–Ausgabemodul eingrenzen. Eine Behebung eines solchen Fehlers bedeutet eine allgemeine Behebung, so dass man sicher sein kann, dass er in dieser Form nicht mehr auftritt.[3]

4.1 Die CSSW— Das Betriebssystem

Die für ein Programm der DPE grundlegende Schnittstelle ist die **Common Service Software** (**CSSW**), die für alle Instrumente auf INTEGRAL gemeinsam entwickelt worden ist. Sie stellt eine Schnittstelle zum Betriebssystem der DPE dar, das sich **Astres** nennt. Dies ist jedoch so gut abgeschirmt, dass der Entwickler einer IASW faktisch nie etwas von Astres sieht und somit die CSSW als Betriebssystem bezeichnen kann. Dennoch ist es gut, von Astres' Existenz

[1]Die Größe des Programms beträgt derzeit etwa 50000 Zeilen, etwa 10000 davon sind echte Anweisungen.

[2]Was sie ja auch ist.

[3]Beispielsweise muss zwischen zwei Kommandos an ein Subinstrument mindestens 10 msec gewartet werden. Dies war anfangs nicht implementiert, konnte aber mit einer *einzigen* Änderung im LSL–Schnittstellenmodul behoben werden.

zu wissen, da es zum einen nicht fehlen darf, wenn die Software in die DPE (mittels PROMS) eingebaut wird. Anderseits gibt es ein paar Eigenheiten der CSSW, die sich auf das Betriebssystem Astres zurückführen lassen.[4] In diesem Kapitel werden zunächst die Ein/Ausgabemöglichkeiten der CSSW vorgestellt. In Kapitel 5 liegt der Schwerpunkt dann auf den Multitaskingfähigkeiten der CSSW.

Hardwaresignale In vielen Fällen muss die Software auf ein bestimmtes **Hardwareereignis** reagieren. Dies wird bei Prozessrechnern üblicherweise mit Hilfe von sogenannten **Interrupts** gelöst. Das sind relativ kleine Unterprogramme, die jederzeit auf ein bestimmtes Hardwareereignis hin angesprungen werden können und dann eine bestimmte Aktion durchführen. Der Name kommt daher, dass der normale Programmablauf dabei kurz unterbrochen ist.

Für ein Datenverarbeitungssystem auf Multitaskingbasis wie das von IBIS ist dieser Ansatz jedoch zu unflexibel, weil im Prinzip jeder Task von einem Hardwareereignis Kenntnis haben sollte. Dies ist mit einer einzelnen Interruptroutine nur schwer zu machen.

Ein Konzept, das mit **Signalen**[5] arbeitet, ist wesentlich leistungsfähiger. Jeder Task kann zunächst auf ein oder mehrere Signale, die ihm von anderen Tasks oder der CSSW geschickt werden, warten. Für einen bestimmten Task können Hardwareereignisse mit solchen Signalen verknüpft werden, d.h. die CSSW schickt beim Auftreten eines Hardwareeereignis ein bestimmtes Signal. Dann kann dieser Task auf solche Ereignisse reagieren, sofern er richtig programmiert ist.[6]
Die IASW verwendet relativ wenig Signale der CSSW, nämlich:

- Ende einer LSL–Übertragung.
- Ende eines HSL–Empfangs.
- Ende einer Analog–Konversion.
- Ende einer Relais–Puls–Übertragung.
- BCP1–Impuls (8–Sekunden–Puls).
- BCP4–Impuls (1–Sekunden–Puls).
- Paritätsfehler auf der LSL.
- Overrun–Fehler auf der LSL.
- FIFO–Überlauffehler auf der HSL.

[4] Dazu gehört etwa, dass im Multitaskingbetrieb an einen suspendierten Task geschickte Signale zwar in der CSSW verloren gehen, nicht jedoch in Astres.

[5] Die offizielle Bezeichnung heißt (laut der CSSW) **event** [20]. Da aber der Begriff *Event* bei IBIS auch für Photonenereignisse benutzt wird, verwende ich lieber die auch sonst in der Literatur anzutreffende Bezeichnung „Signal“ [28].

[6] Natürlich liegt es jetzt an dem Programmierer, ob auf bestimmte Ereignisse schnell genug eine Reaktion erfolgt, denn wenn ein Task längere Zeit sich nicht um die Signale kümmert, werden diese auch verspätet ausgewertet.

Sie seien hier aufgeführt; die genaue Bedeutung wird in Kapitel 5 zusammen mit einer ausführlicheren Beschreibung des Taskmechanismus besprochen. Die volle Auflistung der Signale (bzw. events) findet sich in [20].

Empfang von Telekommandos (TC) Die CSSW kann bis zu sechs Telekommandos über den OBDH–Bus von der CDMU entgegennehmen, bevor ihr interner Puffer überläuft. Sie werden von der IASW über den Aufruf einer CSSW–Prozedur ausgelesen. Wenn kein Telekommando im internen Puffer ist, wartet die Prozedur. Es ist möglich, eine Zeit (Timeout) festzulegen, nach der auf jeden Fall diese Prozedur beendet wird. Das zurückgegebene Telekommando–Feld enthält die Länge des Parameterfeldes, den Typ und Subtyp, alle Parameter, nicht aber den vollen Kopf des Telekommandos. Das macht die Rekonstruktion eines Telekommandos für eine Validierung schwierig.

Abschicken von Telemetriepaketen (TM) Das Abschicken von Telemetriepaketen geschieht durch einen einfachen Prozeduraufruf mit dem Paket als Parameter. Zuvor sollte man allerdings mit einer speziellen Funktion prüfen, ob der CSSW–TM–Puffer noch Telemetriepakete aufnehmen kann. Andernfalls erzeugt die CSSW eine Fehlermeldung.

Zugriff auf die Low speed Serial Line (LSL) Vor einer Verwendung der Low speed Serial Line, muss diese konfiguriert werden (Subinstrumentadresse und Übertragungsgeschwindigkeit). Dann kann mit einem einfachen Prozeduraufruf ein Kommando abgeschickt, ein Parameter gelesen oder beides hintereinander ausgeführt werden. Der CSSW wird dabei ein Puffer übergeben, der das Kommando enthält bzw. in den die Empfangsdaten geschrieben werden.

Nachdem eine solche Operation gestartet ist, sollte bis zu ihrer Beendigung, d.h. auf das schon erwähnte Hardwaresignal gewartet werden. Andernfalls kann ein Fehler auftreten, wenn die Leitung noch verwendet wird.

Schwierig ist dies, wenn die CSSW auf Daten wartet, aber keine geschickt werden. In diesem Falle sollte nach einer bestimmten Zeit die Low speed Serial Line zurückgesetzt werden.[7]

Zugriff auf die HSL Auch hier muss ein Speicherbereich definiert werden, an dessen Adresse geschrieben wird. Das Problem, das bei der LSL nicht auftrat, ist, dass der CSSW eine *physikalische* Adresse für den Puffer übergeben wird, in den die DPE die empfangenen Daten schreibt. Daher muss die Zuordnung zur logischen Adresse dieses Puffers in der IASW sichergestellt werden [20]. Bei der Übertragung sollte bis zum Ende der Operation gewartet werden.

Relaiszugriff Mittels Prozeduraufruf lässt sich an einen der acht Kanäle der DPE ein **Relaispuls** schicken. Bevor ein neuer Puls gesendet wird, sollte geprüft

[7] Im Augenblick muss dabei ein Feature der CSSW verwendet werden: Durch Setzen der LSL auf synchronen oder asynchronen Modus wird in dem LSL–Treiber ein Zurücksetzen provoziert.

werden, ob noch ein Relaispuls verarbeitet wird. Für das Auslesen des Relaisstatus, das nicht mit dem entsprechenden Relaispuls gekoppelt sein muss, ist eine Funktion vorgesehen. Leider gibt es kein Hardwaresignal für das Auftreten einer Relaisstatusänderung, so dass unter Umständen diese Funktion häufig aufgerufen werden muss („Polling").

Analogwerte Für das Einlesen von Analogwerten muss die DPE eine Analog/Digital–Konvertierung starten, die etwa 1 msec dauert. Die CSSW bietet zwei Funktionen an; eine, mit der die Konvertierung gestartet wird, und eine, mit der das Ergebnis ausgelesen werden kann. Vor dem Aufruf letzerer Funktion sollte auf das Hardwaresignal, das das Ende der Konvertierung mitteilt, gewartet werden, da sonst das Ergebnis nicht korrekt ist. Eine Sonderform von Analogwerten ist der Bilevel, der den Analogwert auf einen binären Wert (grösser 128/kleiner 128) einschränkt.

On–Board–Time Gelegentlich wird die an Bord des Satelliten verwendete Zeit (OBT) benötigt. Diese kann mit zwei Funktionen bestimmt werden:

- Exakte Zeitbestimmung: Die Zeit wird in einem Register gelatcht (≫eingefroren≪) und dann ausgelesen.
- Grobe Zeitbestimmung: Die Zeit wird nur aus dem Register ausgelesen. Sie ist dann aber dennoch mindestens auf 125 msec genau bestimmt, da sie mit jedem Task–Takt (RTC) von der CSSW gelatcht wird.

Zusätzlich ist eine Zeitbestimmung des letzten Ein–Sekunden–Pulses möglich. Wie wir sehen werden, können damit die Detektoren synchronisiert werden.

4.2 Die Schnittstellen nach Außen

Wie man sieht, ist durch die CSSW schon eine sehr gut definierte Schnittstelle gegeben, die meist ausreicht. Die IASW muss sie nur dort erweitern, wo aufwendige Protokolle vorgesehen sind. Das betrifft besonders den Zugriff auf die Low speed Serial Line, aber auch einige andere Module, die hier ebenfalls abgehandelt werden.

4.2.1 Gemeinsamer Zugriff auf externe Resourcen

Eines der Hauptprobleme für eine Schnittstelle zur Außenwelt stellt die Vergabe von Resourcen dar. Unter einer **Resource** ist dabei alles zu verstehen, was zum Betrieb eines Systems notwendig, aber nur beschränkt verfügbar ist. Dazu gehört etwa ein Speicherbereich, Rechenzeit des Prozessors oder die Benutzung eines externen Gerätes, welches Aufträge nur nacheinander abarbeiten kann (z.B. Drucker eines UNIX–Clusters). Im Fall der IASW sind die seriellen Leitungen LSL und HSL eine wichtige, häufig benötigte Resource. Obwohl mehrere Subinstrumente bzw. Kanäle angeschlossen sind, darf immer nur eine

Übertragung stattfinden. Gerade dann stellt sich aber die Frage: Wie verhindert man, dass ein Task auf eine serielle Schnittstelle zugreift, wenn noch eine Übertragung stattfindet? Es gibt drei Ansätze:

1. Die anarchistische Lösung: Jeder Task muss sich selber darum kümmern, ob er gerade die serielle Leitung verwenden darf.

2. Die diktatorische Lösung: jeder Task darf nur zu ganz bestimmten Zeiten auf die serielle Leitung zugreifen.

3. Die „sozialdemokratische" Lösung: Jeder Task darf auf die serielle Schnittstelle zugreifen (das ist das demokratische Element), sofern sie nicht von einem anderen Task gesperrt ist (die soziale Komponente). Die Sperrung erfolgt an zentraler Stelle.

Der letzte Vorschlag bietet ein Optimum an Flexibilität und Sicherheit, sofern beim Zugriff bestimmte Spielregeln beachtet werden.

Semaphore Das Sperren eines Zugriffes funktioniert im Prinzip mit einer boolschen Variable, die auf »blockiert« oder »nicht blockiert« stehen kann. Wenn ein Task auf die Low speed serial Line zugreifen möchte, ruft er zuvor eine Prozedur, die `get_lsl` heißt, auf. Ist die Variable auf »nicht blockiert«, so wird sie auf »blockiert« gesetzt, andernfalls wird gewartet, bis sie umgesetzt wird. Nach Benutzung der seriellen Leitung muss die Prozedur `release_lsl` benutzt werden, die die Variable auf »nicht blockiert« setzt. Das sieht so aus:[8]

```
...
get_lsl;
Zugreifen auf die LSL;
release_lsl;
....
```

Intern ist das Schema:

```
VAR lsl_blocked: boolean := FALSE;

PROCEDURE get_lsl
      WHILE lsl_blocked DO
             nix;
      lsl_blocked := TRUE;
END get_lsl;

PROCEDURE release_lsl
        lsl_blocked := FALSE;
END release_lsl;
```

Es gibt zwei wesentliche Probleme bei diesem Ansatz. Zum einen ist er nicht sicher, weil in der Prozedur `get_lsl` der Task nach der Abfrage »while

[8] Die Darstellung ist in Pseudocode.

lsl_blocked≪ und vor dem Setzen der Variable von einem anderen Task höherer Priorität unterbrochen werden kann, der dann ebenfalls eine noch freie serielle Leitung vorfindet und diese unerlaubterweise nutzen könnte. Zum anderen würde ein wartender Task eine große Verschwendung an Rechenzeit bedeuten, was fatal ist, wenn er die höchste Priorität hat und dadurch das ganze System beeinträchtigt. Um diese Schwierigkeiten zu beheben, hat der Informatiker E. W. Dijkstra[9] eine Konstruktion vorgeschlagen, die er **Semaphore** (Zeichenträger) nannte. Sie sieht wie die Prozeduren `get_lsl` und `release_lsl` aus, hat aber zwei entscheidende Unterschiede[10]:

1. Es wird sichergestellt, dass nur ein Task sich in einem **kritischen Bereich** befindet.
2. Bei blockierter Leitung fragt der Task nicht fortwährend den Leitungszustand ab, sondern wartet auf ein Signal von außen.

Dies sieht dann so aus:

```
PROCEDURE get_lsl
        Start Kritischer Bereich;
        IF lsl_blocked THEN
                Ende Kritischer Bereich;
                Warten auf Signal;
                Start Kritischer Bereich;
        END IF;

        lsl_blocked := TRUE;
        Ende Kritischer Bereich;
END get_lsl;

PROCEDURE release_lsl;
        Verschicke Signal an einen Task,
        der auf die LSL wartet;
END release_lsl;
```

In einem kritischen Bereich darf kein Task auf ein Signal warten. Andernfalls sind alle Tasks vom Scheduler ausgeschlossen, da der einzige im kritischen Bereich laufende Task auf Signale von anderen, nicht laufenden Tasks wartet. Aus diesem Grund ist die Prozedur `get_lsl` etwas verschachtelt.

Umsetzung Die Leitungsverwaltung in der IASW ist wie eben beschrieben implementiert. Glücklicherweise kennt die CSSW die Möglichkeit, kritische Bereiche zu definieren; andernfalls wäre es sehr schwierig geworden, Punkt 1 des

[9] Siehe auch [28].

[10] Die Beschreibung der Semaphore von Dijkstra schlug außerdem anstelle der boolschen Variable einen Zähler vor, der die Zugriffsanfragen mitzählt. So wird verhindert, dass die Leitung frei gegeben wird, obwohl noch ein Task auf Zugriff wartet. Im vorliegenden Fall ist dies wegen dem speziellen Message–Konzept der CSSW nicht notwendig, da hierbei keine Signale verloren gehen können. Diese Variante, die eine boolsche Variable zur Blockierung verwendet, wird *binäres Semaphor* genannt [28].

Semaphores zu implementieren.[11] Ein Problem ist die Unkenntnis eines Taskes darüber, ob und welcher andere Task gerade auf die serielle Leitung wartet. Daher wurde das Konzept dahingehend abgewandelt, dass an *alle* Tasks eine Leitung–Frei–Message (ein task–unabhängiges Signal) geschickt wird; innerhalb von `get_lsl` muss dann aber bestimmt werden, ob ein Task tatsächlich Zugriff bekommen hat.[12] Zusätzlich ist für die Low speed Serial Line noch die Möglichkeit (`announce_lsl`) hinzugekommen, die serielle Leitung zu reservieren, d.h. ein (aber nur ein) Task kann bestimmen, dass ihm als nächstes die serielle Leitung zugesprochen wird. Dies ist für den Task der Einzelereignisverarbeitung notwendig, denn dieser sollte nicht längere Zeit untätig auf ein Ereignis warten. Andernfalls besteht die Gefahr, dass die Puffer in der HEPI überlaufen und Daten verloren gehen. Für Fragen der genauen Implementierung sei auf den Quellkode im Modul LINE_MANAGER verwiesen.

»Spielregeln« Natürlich funktioniert das Schema nur, wenn alle Programmteile die Prozeduren `get_lsl` und `release_lsl` beim Zugreifen auf eine serielle Leitung verwenden, da andernfalls Schnittstellenfehler wegen Doppelbelegung auftreten können. Fataler ist ein Vergessen von `release_lsl`, weil dadurch die serielle Leitung, und damit auch das ganze System blockiert bleibt. Es sind daher folgende Kontrollen implementiert worden:

- Wenn das Hardwaresignal »Übertragungsende« von der CSSW nach dem Schicken oder Lesen der Daten auftritt und die serielle Leitung ist nicht blockiert (das Blockieren ist also vergessen worden), so wird eine Ereignismeldung erzeugt.

- Wenn ein Task, der die Leitung blockiert, erneut `get_lsl` aufruft, wird ebenfalls eine Ereignismeldung abgeschickt, da offenbar der Aufruf `release_lsl` vergessen wurde (es sei denn, die Software hat ganz gravierendere Mängel im Design).

Insofern kann bei vernünftigem Testen garantiert werden, dass das Semaphore–Konzept sicher funktioniert.

4.2.2 Die Low-Speed-Serial-Line

Der Zugriff auf die Low speed Serial Line geschieht in der IASW in mehreren Ebenen.

Basisoperationen Die unterste Ebene der LSL–Schnittstelle fasst im Modul LSL_INTERFACE die CSSW–Befehle für die Benutzung der LSL zusammen, indem sie die LSL initialisiert und eine Rücksetzoperation anbietet. Letztere

[11] Es gibt Wege, um Semaphores ohne kritische Bereiche zu implementieren [28], aber diese sind nicht durch strukturelle Einfachheit ausgezeichnet.

[12] Das bedeutet indirekt, dass der Task mit der höchsten Priorität als erster auf die serielle Leitung zugreifen darf.

schaltet auf asynchrone Übertragung, wodurch CSSW–intern ein Zurücksetzen provoziert wird. Bei der Initialisierung findet eine Umsetzung der LSL–Adressauswahl statt.[13]

Detektor–Schnittstelle Für die normale Benutzung der Low speed Serial Line ist es wünschenswert, wenn einer Prozedur das Kommando abgegeben bzw. ein Feld für den Empfang bereitgestellt wird und die Übertragung innerhalb der Prozedur stattfindet. Dies ist mit den Prozeduren `Send_CMD` bzw. `Read` des Modules DETECTOR_INTERFACE möglich. Die Parameter/Empfangsdaten können beliebige Länge haben, was die Programmiersprache ADA mit unbeschränkten Feldern unterstützt. Aus der Größe des Feldes wird dann die Länge des Kommandos bzw. der zu empfangenden Daten bestimmt.[14] Innerhalb dieser Operationen findet allerdings *keine* Blockierung der LSL statt. Dem Benutzer ist dadurch die Freiheit gegeben, ob die LSL mit der Prozedur `get_lsl` oder `announce_lsl` blockiert werden soll.

In diesem Modul befindet sich auch die Behandlung aller Hardwarefehler der LSL, wobei bei einem Versagen der Operation oder einem Timeout eine Ereignismeldung abgeschickt wird. Diese enthält die Information darüber, was für ein Kommando aus welchem Task mit welcher Nummer abgeschickt wurde. Der Erfolg der Operation wird dem Benutzer mit einem boolschen Wert angezeigt.

Bedeutsam ist noch die ≫10 msec–Klausel≪: zwischen zwei Kommandos an ein gleiches Subinstrument müssen mindestens 10 Millisekunden verstreichen [33]. Dazu wird geprüft, ob das letzte Kommando an das gleiche Subinstrument gegangen ist, und ob dies vor weniger als 10 Millisekunden war. In diesem Fall wird maximal 135 Millisekunden (rund ein RTC) gewartet.[15]

≫Very High Level≪ Von der IASW sollten nicht beliebige Kommandos an die Subinstrumente geschickt werden. Daher ist für jedes Subinstrument ein Modul als Befehlsschnittstelle vorgesehen, welches für Kommandos der IASW gedacht ist. Externe Kommandos, die via Telekommando TC (5,3) das Subinstrument erreichen sollen, verwenden das Modul `detector_interface`. Der Name des Moduls entspricht dem des Subinstrumentes. In den jeweiligen Modulen befindet sich eine Liste gültiger Kommando–Identifier (als Aufzählungstyp), die Grösse der Housekeepingdaten und Operationen zum Schicken eines Kommandos bzw. einer Leseanfrage. Alle Befehle, die an ein bestimmtes Subinstrument

[13] Das ist insofern nicht trivial, weil für den 1750A-Prozessor Bit 0 eines Words das höchstwertige Bit, Bit 15 das niederwertigste Bit ist, aber für das Adressregister des LSL auf der DPE es genau umgekehrt ist, so dass eine Bit–Spiegelung vorgenommen werden muss.

[14] Für eine flexible und speichersparende Implementierung empfiehlt es sich, die Parameterfelder lokal in einer Prozedur, d.h. auf dem Stapelspeicher anzulegen.

[15] Dies ist die kleinste Zeiteinheit, die im Multitasking mit Warten verbracht werden kann. Eine andere Möglichkeit wäre, periodisch die Zeit abzufragen, was aber für 10 Millisekunden pro Kommando eine große Verschwendung an Rechenzeit bedeutet. Eine weitere Möglichkeit ist die Verwendung eines Hardware–Zeitgebers (Timer A), der ein beliebig konfigurierbarer „Wecker“ ist. Die Schwierigkeit hierbei ist, dass jeder Task mit einem Hardwaresignal verbunden werden müßte.

gehen, werden gezählt.[16]

Das Modul HEPI enthält noch weitere Operationen. Mit einem elektrischen Puls an eine Relaisleitung der DPE kann die HEPI synchronisiert werden. Die Synchronisation besteht aus folgenden Schritten:

- Warten auf ein 1–Sekunden–Hardwaresignal (BCP 4) von der CDMU.
- Synchronisationsbefehl an die HEPI schicken. HEPI setzt dann mit dem nächsten BCP 4–Signal die interne Zeit für Einzelereignisse auf Null.
- In der IASW erneut auf BCP 4 warten.
- An–Bord–Zeit des letzten BCP 4–Signals auslesen. Diese Synchronisationszeit wird von der IASW als Zeitnullpunkt für alle HEPI–Daten verwendet.

Häufig ist es nötig, im HEPI–Statusregister nur einige Bits zu ändern. Damit dazu nicht das Statusregister ausgelesen werden muss, um den aktuellen Inhalt zu bestimmen, wird jeder Zugriff auf dieses Register, das über das HEPI–Modul geht, in diesem Modul gespeichert.[17] Der aktuelle Statuswert kann mit der Prozedur `Hepi_Status_Shadow` gelesen werden. Schliesslich kann über die Funktion `HBR_A_Data_Available` und `HBR_B_Data_Available` festgestellt werden, ob aus der HEPI Daten ausgelesen werden können. (Siehe auch Kapitel 5.5.2 und 5.6.1.)

4.2.3 Die High-Speed-Serial-Line

Der Zugriff auf die **High speed Serial Line** ist auf unterster Ebene im Modul HSL_INTERFACE implementiert. Der Zugriff auf höherer Ebene ist komplex, so dass für diese Zugriffe eigene Tasks (HBR–A–Task und Read–Histogramm–Task) verwendet werden. Die Tasks werden in den Kapiteln 5.5.2 und 5.6.1 beschrieben.

4.2.4 Die Analoge Schnittstelle

Die Schnittstelle des Analog/Digitalwandlers im Modul ANALOG_IF ist verhältnismäßig einfach gestaltet. Es genügt, die Funktion `Fetch_Analog_Val` mit gegebenem Analogkanal aufzurufen, die den Analogwert via CSSW anfordert. Die Funktion wartet auf das Ende der Konversion, das über das Hardware–Signal »End Of Analog Acquisition« mitgeteilt wird. Vorraussetzung ist, dass der Task mit diesem Signal verbunden ist. Wenn dieses Signal nach 125 msec (ein RTC) nicht erscheint, wird eine Timeout–Ereignismeldung abgeschickt.

[16] Das betrifft natürlich nur die Kommandos, die diese Schnittstelle auch tatsächlich verwenden, was für externe Telekommandos vom Typ TC(5,3) nicht der Fall ist. Daher werden nur Kommandos gezählt, die die IASW abschickt.

[17] Der Nachteil ist, dass das HEPI–Modul ja mit einem externen Kommando umgangen werden kann. Da aber die Bits nur im Science–Mode geändert werden müssen, anderseits aber gerade im Science–Mode solche Kommandos gesperrt sind, sollte dieses Vorgehen keine Schwierigkeiten machen. Bei Eintritt in den Science–Mode wird durch die Neusynchronisation das HEPI–Statusregister gelöscht.

4.2.5 Die Telemetrie–Schnittstelle

Alle Telemetriepakete der IASW werden der Prozedur `Put_TM` im Modul TM_INTERFACE übergeben. Ausgenommen sind die Standard–Housekeeping–Pakete der IASW, für die die CSSW eine eigene Schnittstelle bereitstellt. Die Funktion prüft zunächst, ob das Telemetriepaket noch Platz im Telemetriepuffer der CSSW hat. Wenn nicht, so wird es mit einer Ereignismeldung verworfen, andernfalls der CSSW übergeben.

In dem Modul ist weiterhin eine Liste aller Typen von Telemetriepaketen definiert, die sich durch APID, Typ und Subtyp unterscheiden. In einer Telemetrietabelle ist für jeden Telemetrietyp eine spezifische APID, die Priorität des Telemetriepaketes und eine obere wie auch untere Schwelle definiert.

Eine Heuristik, die in diesem Modul implementiert ist, entscheidet bei erhöhtem Telemetrieaufkommen, welche Telemetriepakete nicht erzeugt und abgeschickt werden dürfen. Dazu können alle Prozesse der IASW über die Funktion `Must_I_Process` abfragen, ob die Erzeugung eines Telemetriepakets angehalten werden muss. Innerhalb des Moduls werden für jeden Telemetrietyp die Schwellwerte aus der Telemetrietabelle untersucht. Ist der Telemetriepuffer voller als die obere Schwelle eines bestimmten Telemetriepakettyps, so ist das Abschicken nicht erlaubt. Sinkt er unter die untere Schwelle, so ist das Abschicken gewährt. Durch die Hysterese der Schwellen soll ein laufendes An/Abschalten vermieden werden.

Da die Abfrage des CSSW–Telemetriepuffers zeitaufwendig (etwa 108 μsec) ist, vergleicht man die Schwellwerte nicht mit dem tatsächlichen, sondern einem interpolierten Pufferstand, der nach 200 Abfragen aufgefrischt wird. Die Interpolation kann anhand der gesendeten Telemetriepuffer bestimmen, wie voll der Telemetriepuffers maximal ist. Eine genaue Beschreibung findet sich in [27].

4.2.6 Die On–Event–Schnittstelle

Wenn die IASW **Ereignismeldungen** (On–Event–Messages — OEM) erzeugt, muss sie eine Schnittstelle im Modul OEM_INTERFACE verwenden. Die Zahl aufgetretener Ereignismeldungen erscheint im Housekeeping–Paket der IASW und wird in diesem Modul bestimmt.

Normalerweise werden Ereignismeldungen mit einer allgemeinen Prozedur namens `Report_OEM` abgeschickt. Ihr wird ein Bezeichner, der die Ereignismeldung klassifiziert, und drei Felder, die nähere Information zur Ereignismeldung liefern, übergeben. In Tabelle 4.1 sind alle Ereignismeldungen der IASW festgehalten, wobei die Zahl der relevanten Parameter (Feld eins bis drei) variieren kann.[18]

[18] Die Klasse/Identifier–Aufteilung ist innerhalb des Moduls definiert und kann sich noch ändern. Stand: 29.3.1999.

Klasse	ID	Beschreibung	Parameter
0	128	Veto ausgeschalten	0
0	129	Kontext gesichert	3 Context Save Flags
0	130	Kontext wiederhergestellt	4 Context Restore Flags
0	131	In den Standby–Mode gegangen	2
0	132	Hepi resynchronisiert	1
1	128	Exception ist aufgetreten	0 Except.–Nr. Task–ID Exception-Adresse
1	130	Überlauf an Telemetriepaketen	0 0 Sequenz–Zähler
1	131	Einzelereignisse mit negativer Zeitfolge	Einzelereignis–Typ (S-1,S-3, S-4) LSW der Zeit (danach) LSW der Zeit (davor)
2	128	Timeout bei Analogkonvertierung	0 Analogkanal$\times$256
2	129	Timeout bei LSL–Übertragung	MSBit: Hin/Rückweg; Task–ID CID Adresse Sequenz–Zähler
2	130	Timeout bei HSL–Übertragung	Task–ID HBR A,B (0,1) Block–Adresse (nur HBR–B)
2	131	Versagen der LSL–Übertragung (high level)	MSBit: Hin/Rückweg; Task–ID CID Adresse Sequenz–Zähler
2	132	Versagen der HSL–Übertragung (high level)	Task–ID HBR A,B (0,1) Block–Adresse (nur HBR–B)
2	133	Fehler bei LSL–Übertragung	0 0 - Paritätsfehler; 256 - Overrun
2	134	Fehler bei HSL–Übertragung (FIFO–Überlauf)	0
2	135	Fehler bei Puls–Kommando	0 Puls–Kanal $\times$256
2	136	Versagen beim Veto–ausschalten	0
2	137	Timeout beim Schicken von Histogramm–Telemetrie	Telemetrie–Typ Sequenz–Zähler

<table>
<tr><th>Klasse</th><th>ID</th><th>Beschreibung</th><th>Parameter</th></tr>
<tr><td>3</td><td>128</td><td>TC: Ungültiges Typ/Subtyp–Feld</td><td><table><tr><td colspan="2">0</td></tr><tr><td>Typ/Subtyp</td><td>Seq.Zähler</td></tr></table></td></tr>
<tr><td>3</td><td>129</td><td>TC: Ungültige TID/FID</td><td><table><tr><td colspan="2">1</td></tr><tr><td>Typ/Subtyp</td><td>Seq.Zähler</td></tr><tr><td>TID</td><td>FID</td></tr></table></td></tr>
<tr><td>3</td><td>130</td><td>Telekommando–Timeout</td><td>0</td></tr>
<tr><td>3</td><td>132</td><td>TC: Ungültiger Parameterwert</td><td><table><tr><td colspan="2">0</td></tr><tr><td>Typ/Subtyp</td><td>Seq.Zähler</td></tr><tr><td colspan="2">Parameter–Wert</td></tr></table></td></tr>
<tr><td>3</td><td>133</td><td>TC: Ungültige Parametergröße</td><td><table><tr><td colspan="2">0</td></tr><tr><td>Typ/Subtyp</td><td>Seq.Zähler</td></tr><tr><td colspan="2">Parameter–Länge (Byte)</td></tr></table></td></tr>
<tr><td>3</td><td>134</td><td>TC: Versagen beim Ausführen</td><td><table><tr><td colspan="2">0</td></tr><tr><td>Typ/Subtyp</td><td>Seq.Zähler</td></tr></table></td></tr>
<tr><td>3</td><td>135</td><td>TC: Ungültiger Mode</td><td><table><tr><td colspan="2">0</td></tr><tr><td>Typ/Subtyp</td><td>Seq.Zähler</td></tr><tr><td colspan="2">IASW–Mode</td></tr></table></td></tr>
</table>

Tabelle 4.1: *Die Ereignismeldungen der IASW*

Die Ereignismeldung von Exceptions meldet einen Softwarefehler. Ihre Bedeutung ist in Kapitel 5.8.1 ausführlich beschrieben.

Diese Ereignismeldung enthält eine Information über die Stelle , an dem die Exception aufgetreten ist, bzw. ihre Ursache in Form einer Exception–Nummer. Die Exception–Nummern und die zugehörigen Modulnamen bzw. Fehler sind in Tabelle 4.2 aufgelistet.

0	HBR A Task	38	Diagnostic Process CsI
1	HBR B Task	39	Diagnostic Process CdTe
2	Read Histogram Task	40	Diagnostic Reset
3	Line Manager Task	41	Diagnostic Close
4	(Main Control Task)	42	Transparent Process80
5	Main Task	43	Transparent Process160
6	TC Task	44	Transparent Reset
7	HK Task	45	Transparent Close
8	TM Interface	46	E-Corr1
9	TM Must-I-Process	47	E-Corr2
10	S1 Process	48	E-Corr3
11	S1 Reset	49	E-Corr4
12	S1 Close	50	E-Corr5
13	S2 Process	51	Energy Correction
14	S2 Reset	52	Analog I/F Analog
15	S2 Close	53	Analog I/F Bilevel
16	S3 Process Multi	54	HEPI Context
17	S3 Process Single	55	PICSIT Context
18	S3 Reset	56	ISGRI Context
19	S3 Close	57	VETO Context
20	Compton Select	58	Table Handler
21	S4 Process Multi	59	Science HK
22	S4 Process Single	60	Detector I/F
23	S4 Reset	61	Line Manager LSL Not Blocked
24	S4 Close	62	Line Manager HSL Not Blocked
25	S5 Process	63	Line Manager HSL Nested
26	S5 Reset	64	Line Manager LSL Nested
27	S5 Close	64	S5 Addresses Noncontinuous
28	S6 Process	65	S6 Addresses Noncontinuous
29	S6 Reset	66	S7 Addresses Noncontinuous
30	S6 Close	67	HEPI Register II ungültig
31	S7 Process Single		
32	S7 Process Multi		
33	S7 Reset		
34	S7 Close		
35	S8 Process		
36	S8 Reset		
37	S8 Close		

Tabelle 4.2: Nummern der Exception–Bezeichner

Kapitel 5

Die Tasks der IASW

Die IASW ist in mehrere, weitgehend unabhängig voneinander arbeitende Programmteile aufgeteilt. In Tabelle 5.1 sind diese Progammteile mit ihren Task–Namen aufgeführt. Vor einer genauen Beschreibung wird auf Multitasking im allgemeinen und die Form, die es in der DPE hat, eingegangen.

Task–Name	Beschreibung	Task –ID	Priorität
Main–Task	Übernimmt den Hauptteil der Datenverarbeitung von Einzelereignissen, repräsentiert die Modi der IASW.	1	3
TC–Task	Wartet auf Telekommandos und führt sie aus.	2	11
HK–Task	Sammelt alle 8 Sekunden Housekeeping–Daten.	4	12
Histogram–Task	Kontrolliert die Integration von Histogrammen.	5	10
Read–Histogram–Task	Liest Histogramme aus.	6	4
HBR A–Task	Liest Einzelereignisse ein.	3	8
Line–Manager–Task	Überwacht den Zugriff auf die seriellen Leitungen.	-	–[1]

Tabelle 5.1: Die Programmteile der IASW und ihre Tasknamen

5.1 Multitasking — Wie und Wieso

Die Aufgaben der IASW müssen zum Teil nebeneinander ausgeführt werden. So muss sie beispielsweise in der Lage sein, Telekommandos abzuarbeiten und dennoch alle acht Sekunden Housekeeping–Daten abzuschicken. Dies sind Anforderungen, wie sie normalerweise an Prozessrechner gestellt werden.

Aufgabenverteilung Wie schon im Zusammenhang mit den Hardwaresignalen in Kapitel 4.1 angedeutet wurde, gibt es mehrere Möglichkeiten, wie Aufgaben parallel verarbeitet werden können.

Interrupts Die schnellste Reaktion auf ein äußeres Ereignis erfolgt über ein elektrisches Signal der Hardware, das eine Prozedur aufruft, die den bisherigen Programmfluss unterbricht und die gewünschte Aufgabe erledigt. Dies wird als Unterbrechungsbehandlung oder **Interrupt** bezeichnet.

Problematisch wird es, wenn die Abarbeitung der Aufgabe lange dauert und viele verschiedene Interrupts gleichzeitig auftreten, weil dann die Reaktionszeit größer wird und unter Umständen Interrupts verloren gehen können. Da Interrupts schon vom Konzept her unvorhersehbar auftreten können, ist eine Untersuchung der Zuverlässigkeit relativ schwierig.[2]

Polling Wenn das Programm in einer Schleife periodisch abfragt, welche Aufgaben zu erfüllen sind, so spricht man vom **Polling**[3]. Das Polling ist sehr zuverlässig (jede Aufgabe wird strikt nacheinander erledigt), allerdings kann die Reaktionszeit sehr groß werden, wenn einzelne Aufgaben lange dauern. Daher ist es häufig notwendig, eine einzelne Aufgabe in mehrere Teile aufzuteilen. Das bedeutet aber, dass größere Programme sehr unflexibel werden, denn das Trennen einer Aufgabe in mehrere Teile kann für den Programmierer mühsam und zeitaufwendig sein und muss für jede größere Änderung wiederholt werden.

Multitasking Das Multitaskingkonzept bewegt sich etwa in der Mitte zwischen den ersten beiden Ansätzen. Die Idee dabei ist, dass mehrere Programmteile, eben die **Tasks**, gleichzeitig gestartet werden. Jeder Programmteil kann eine gewisse Zeit laufen, dann wird der sogenannte **Scheduler**[4] aktiviert, der entscheidet, welcher Task von nun an weiter laufen darf. Natürlich darf kein Task jemals ein Ende erreichen, weil dann der Task in einen undefinierten Zustand gehen würde.[5] Deshalb sind alle Tasks als Endlosschleifen programmiert.

Es gibt einige Varianten, wie dieser Taskwechsel vor sich geht. Der wesentliche Unterschied ist, ob jeder Task selbstständig den Scheduler an geeigneten Stellen aufrufen muss (**nicht–preemptives Multitasking**), oder ob der Scheduler einen Task jederzeit anhalten darf (**preemptives Multitasking**). Ersteres ist mit einer angenehmen Variante des Pollings gleichzusetzen, sofern der Scheduler alle Tasks nacheinander aufruft. Jede Aufgabe zerteilt sich selbst durch den Aufruf des Schedulers. Denn bei einem solchen Aufruf werden alle aktuellen Werte gesichert, und wenn der Scheduler diesen Task wieder aufruft, läuft er genau an der Stelle weiter, an der er vorhin aufgehört hatte. Es muss

[2] Die Zuverlässigkeit kann mit einem Prioritätsschema von Interrupts erhöht werden.

[3] Der Begriff kommt daher, dass das Programm »wählt«, welche Aufgabe als nächstes zu erledigen ist.

[4] Der aus dem Englischen kommende Begriff bezeichnet ursprünglich die Fahrplanerstellung im Zugverkehr.

[5] Beim Multitasking der CSSW erfolgt ein Rücksprung zum Initialisierungsprogramm, das sein Ende erreichen würde und damit einen Haltepunkt erreicht.

dann nur darauf geachtet werden, dass genügend Scheduler–Aufrufe im Task–Programm verteilt sind. Das Problem ist nur, dass alle Tasks sich kooperativ verhalten müssen und in keinem Fall zu lange laufen dürfen. Unter Umständen kann so ein kleiner Softwarefehler eines Tasks massive Störungen hervorrufen.

Das Preemptive Multitasking legt mehr Gewicht auf die instantane Unterbrechbarkeit, die den Interrupts eigen ist. Der Scheduler empfängt (wie schon in Kapitel 4.1 beschrieben) Hardwaresignale, die als Signal an einen Task weitergeleitet werden. Dieser Task arbeitet dann quasi als Interrupt–Prozedur. Zusätzlich können die Tasks sich auch gegenseitig Signale schicken und sich dadurch gegenseitig synchronisieren.

Weitere Feinheiten ergeben sich aus der Art und Weise, wie der Scheduler die Rechenzeit unter den Tasks verteilt. Bekannt ist das Round–Robin, bei dem alle Tasks der Reihe nach angesprochen werden, und das **Prioritätsgesteuertes Multitasking**, das immer den Task mit der höchsten Priorität laufen lässt. Siehe auch [28] und [25].

Zeiteinteilung Gerade in der letzten Variante stellt sich die Frage, wie Tasks mit geringerer Priorität einmal an die Reihe kommen können. Die Antwort ist einfach: jeder Task, der auf Signale wartet, wird vom Scheduler nicht vor dem Eintreffen eines Signals aktiviert, so dass auch Tasks geringer Priorität laufen können. Daher ist in den meisten Multitaskingsystemen ein Zeittakt vorgesehen, der in regelmäßigen Abständen ein Signal schicken kann, auf das die Tasks warten können. In manchen Systemen[6] wird mit jedem Zeittakt die Priorität geändert, so dass jeder Task an die Reihe kommen kann.

Nachteile Es ist allerdings schwierig, beim prioritätsgesteuerten Multitasking abzuschätzen, welcher Task wann und wie lange ausgeführt wird. Ein Task mit hoher Priorität, der es versäumt, ab und zu auf ein Signal zu warten, kann das System erheblich beeinträchtigen.

Gravierender sind Verklemmungen oder **Dead Locks**. Gemeint ist damit die Möglichkeit, dass Task A auf ein Signal von Task B wartet, während Task B nicht ohne ein Signal von Task A weiterläuft. Beide Tasks werden ewig warten.

Auf das Problem der Resourcenverwaltung ist schon in Kapitel 4.2.1 eingegangen. Dort lag das Problem in der Schwierigkeit, Tasks wirksam von einer Mehrfachbenutzung der seriellen Schnittstellen abzuhalten. Es liegt in der Verantwortung des Programmierers das System so zu gestalten, dass diese Fehlerquellen nicht auftreten können.

Implementierung in der CSSW Das Multitasking der CSSW hat folgende Eigenschaften:

- Es ist präemptiv und prioritätsgesteuert.
- Haben zwei Tasks die selbe Priorität, wechseln sie sich wie beim Round–Robin ab. Die Priorität wird bei der Initialisierung festgelegt und bleibt konstant.

[6] z.B. UNIX.

- Der **Task–Kontext**, der den Zustand eines Tasks bestimmt, beschränkt sich auf den Stapelspeicher, die Registervariablen und den aktuellen Adresszustand.

  Korrekterweise handelt es sich hier um **Multithreading**, bei dem die Speicherbereiche der Tasks nicht getrennt sind und verschiedene Tasks die gleiche Prozedur aufrufen können. [25]. Solche Prozeduren, die im wesentlichen für Schnittstellen verwendet werden, müssen daher **reentrabel** sein, d.h. ohne Schaden gleichzeitig von mehreren Tasks aufrufbar sein.

Die CSSW[7] enthält Operationen, mit denen eine Prozedur als Task deklariert werden kann, und mit denen Tasks generell gestartet und gestoppt werden können. Jedem Task ist eine **Task–ID** zugewiesen, die abfragbar ist. Sie wird jedem Task bei der Initialisierung zugewiesen und ist sehr nützlich für Fehlerberichte (sie erscheint in einigen Ereignismeldungen) und für eine Schnittstellenkontrolle (es wird vermerkt, welcher Task gerade welche serielle Leitung sperrt).

Signale der CSSW Jeder Task kann auf Signale warten und kann über eine 16–Bit–Variable anhand der gesetzten Bits ermitteln, welche Signale aufgetreten sind. Dazu gehören die in Kapitel 4.1 schon beschriebenen Hardwaresignale, aber zusätzlich auch **Software–Signale**[8], die von anderen Tasks geschickt werden können.

Zyklische Tasks Wenn ein Task periodisch eine bestimmte Aufgabe erledigen soll, so ist es wünschenswert, wenn er dazu mit einem Zeittakt synchronisiert wird, da so die Rechenzeit gut verwaltet werden kann.

Die CSSW erzeugt einen Zeittakt in sogenannten **Real Time Clocks** (RTC) von 125 msec. Sie enthält außerdem einen Task mit sehr hoher Priorität, der jedem Task mit einer **Periode** von Vielfachen eines RTC ein Signal schicken kann. Der Task muss dazu die Periode bei der CSSW deklarieren. Dadurch ist der Task **zyklisch** gemacht worden, sofern er regelmäßig (vor Beginn der nächsten Periode) auf das Zeitsignal wartet.

Unabhängig von der von ihm verbrauchten Rechenzeit läuft der Task dann mit einer kontrollierbaren Reaktionszeit. Komplexe Tasks sollten zyklisch sein, weil es dann erheblich einfacher ist, Ordnung in den Task–Ablauf zu bringen. Zunächst läuft erst der Task mit der höchsten Priorität, wartet auf sein Zeitsignal, der zweite Task kommt an die Reihe usw., und im nächsten Zyklus (der durch das kleinste gemeinsame Vielfache der Einzelperioden definiert ist) geht das Spiel von vorne los. Probleme gibt es, wenn ein Task länger als seine definierte Periode läuft oder wenn ein zyklischer Task auf ein anderes als das Zeitsignal warten muss.

Zusätzlich bietet die CSSW Sonderform der Signale, die sogenannten **Messages**. Der Unterschied zu Signalen ist, dass Messages an alle Tasks geschickt

[7] Eigentlich das Betriebssystem Astres.

[8] auch Software–Events genannt[20].

werden können (Signale werden immer an einen konkreten Task geschickt), und dass die Zahl möglicher Messages nicht auf 16 beschränkt ist.

Das alles bedeutet einen großen Freiraum für die Form der IASW, und es bedarf erheblicher Sorgfalt, damit der Systemablauf nicht zu komplex und damit unüberschaubar wird.

Entry–Point Für jeden Task gibt es einen **Eintrittspunkt**, mit dem er gestartet wird. In der IASW ist dieser durch eine Prozedur festgelegt, deren Adresse bei der Initialisierung der CSSW bekanntgegeben werden muss. Diese Initialisierung findet in dem Modul TASKS statt, in dem auch die Startprozeduren jedes IASW–Tasks definiert sind.

5.2 Der Main-Task

Der **Main-Task** übernimmt die wesentliche Datenverarbeitung, vom Einlesen der Einzelereignisse bis zum Abschicken der Telemetrie an die Bodenstation. Die Bearbeitung ist von den Modi abhängig, in denen sich die IASW befindet.

Der Eintrittspunkt des Main–Tasks besteht aus der Prozedur `Start`, die sich in dem Modul MODE_DISPATCHER befindet. Diese Prozedur besteht aus einer Schleife, die, unterbrochen mit einem Warten auf den nächsten Zeittakt, beständig fragt, welcher Modus der IASW ausgeführt werden soll. Der Modus wird von dem Telekommando–Task gesetzt und wird durch eine Variable (`Running_Mode`) repräsentiert. Der Modus ist mit einer Funktion für alle Softwareteile lesbar. Jeder aufgerufene Modus, der als Prozedur verschiedener Module implementiert ist, wird beendet, indem die Prozedur `Stop` des jeweiligen Moduls aufgerufen wird. Der Aufruf erfolgt, wenn in den Standby–Mode gewechselt wird.

5.2.1 Standby-Mode–Automatismus-Behandlung

Die Schleife, die in der Prozedur `Start` des Moduls MODE_DISPATCHER abläuft, stellt den **Standby–Mode** dar, der der Grundzustand der IASW ist. Alle Aufgaben, wie Parameter ändern oder Housekeeping auszulesen geschehen in anderen Tasks. Der Main–Task übernimmt im Standby–Mode folgende Aufgaben

- Sichern bzw. Wiederherstellen der Subinstrument–Kontexte.

- Abschalten der Hochspannung von VETO.

Die Information, wann so etwas zu geschehen hat, wird von dem Modul BCPK_HANDLER, das alle Broadcast–Pakete verwaltet mit den Funktionen `Save_Context` bzw. `Veto_Off`[9], bereitgestellt. Bei einer solchen Aktion wird der augenblickliche Zustand gespeichert (Kontext gesichert bzw. VETO–Hochspannung ausgeschaltet), damit die Aktion nur einmal während des Wechsels stattfindet.

[9]Siehe auch Kapitel 5.3.3.

Kontextübertragung Der Subinstrument–Kontext kann durch Aufruf der Prozedur `Download_Context` im Modul INSTRUMENT_CONTEXT in die DPE gesichert und der Prozedur `Upload_Context` in den jeweiligen Subinstrument wiederhergestellt werden. Innerhalb dieser Prozeduren wird zunächst über den Wert des ECR[10] geprüft, ob das entsprechende Subinstrument überhaupt angesprochen werden darf. Dann werden die Subinstrumente in einen Zustand gebracht, der das Sichern bzw. Wiederherstellen eines Kontextes erlaubt. Schließlich findet die eigentliche Übertragung mit den Prozeduren `Upload` bzw. `Download` aus den Modulen HEPI_CONTEXT_TABLE, ISGRI_CONTEXT_TABLE, PICSIT_CONTEXT_TABLE und VETO_CONTEXT_TABLE statt, die auch eine Prüfsummenkontrolle durchführen. Die Kommandos, die von der IASW verwendet werden, sind in Tabelle 5.2 dargestellt. Zum Teil handelt es sich auch um spezielle Kommandos der Subinstrumente, die in den entsprechenden Dokumenten beschrieben sind.

Sub–Instr.	Kontext sichern	CID	Kontext restaurieren	CID
HEPI	—		≫Kontext schreiben≪	9
ISGRI	≫Goto Standby≪	21	≫Kontext schreiben≪	26
	≫Kontext lesen≪	30	≫Kontext anwenden≪	22
	≫Prüfsumme holen≪	29	≫Prüfsumme holen≪	29
PICsIT	≫Gehe in Maintenance Mode≪	243	≫Gehe in Nominal Mode≪	242
	≫Kontext lesen≪	160	4 Sekunden Warten	
	≫Prüfsumme holen≪	251	≫Kontext schreiben≪	224
			≫Prüfsumme holen≪	251
			≫Kontext anwenden≪	229
VETO	≫Gehe in Standby Mode≪	1	≫Kontext schreiben≪	224
	≫Kontext lesen≪	9	≫Prüfsumme holen≪	251
	≫Prüfsumme holen≪	251		

Tabelle 5.2: Kommandos für Kontextbehandlung

Ist das Sichern eines bestimmten Kontextes erfolgreich und die Prüfsumme in DPE und Subinstrument gleich, so wird im Modul INSTRUMENT_CONTEXT ein Bit der **Context Save Flags** (CSF) gesetzt, das in dem Housekeeping–Paket den Erfolg der Übertragung wiederspiegeln soll. Analog melden die **Context Restore Flags** (CRF) die Wiederherstellung eines bestimmten Kontexts im Subinstrument. Allerdings findet eine Wiederherstellung des Kontexts nur dann statt, wenn die Sicherung des Kontextes erfolgreich war. Der Vorgang der Kontextübertragung wird auch in Kapitel 5.3.1 angesprochen.

[10]Das **Electronics Configuration Register** im Modul ECR definiert, welches Subinstrument überhaupt eingeschalten ist. Es wird mit einem Telekommando gesetzt. Siehe auch Kapitel 3.1.1.

Abschalten von VETO–Hochspannung Wenn die Strahlenbelastung zu hoch ist, muss von VETO die Hochspannung des Photomultiplier abgeschalten werden. Damit dies sicher geschieht, wird an VETO der Befehl »Goto Standby« geschickt, der damit unter anderem das gewünschte Abschalten bewirkt.[11] Dieser Befehl ist als eigene Prozedur (`Switch_Veto_Off`) im Modul VETO definiert.

5.2.2 Science-Mode–Einzel-Event-Bearbeitung

Initialisierung Wenn über ein Telekommando an den MODE_DISPATCHER der Start des Science–Mode angeordnet wurde, wird zunächst die Prozedur `Start` des Moduls SCIENCE_MODE aufgerufen. In ihr findet die Bearbeitung der Einzelereignisse statt.

Zuerst werden alle Bearbeitungsroutinen (S1 bis S4 und S8) und die Ereigniszähler in einen definierten Anfangszustand gebracht (Prozedur `Reset_Science_Functions`). Als nächstes wird HEPI synchronisiert, wobei die Zeit der Synchronisation festgehalten wird. Sie ist über die Funktion `Get_Last_Sync_Obt` des Moduls HEPI verfügbar, siehe auch 4.2.2. Dann werden aus dem Modul SCIENCE_FUNCTION gemäß dem Satelliten–Status (Pointing–Mode oder Slew–Mode) die aktiven wissenschaftlichen Funktionen bestimmt und das HEPI–Statusregister gesetzt. Dabei dürfen nur die relevanten Bits verändert werden. Dazu wird auf die Funktion `Hepi_Status_Shadow` in Modul HEPI, die eine Kopie des Statusregisters zurückgibt, zurückgegriffen. Schließlich wird das HEPI–Register Typ II via Low speed Serial Line an HEPI geschickt. Nach diesen Schritten ist die HEPI bereit, wissenschaftliche Daten zu liefern.

Start der Histogrammfunktion Wenn die HEPI initialisiert ist, können auch Histogramme gebildet werden. Dazu übergibt der Main–Task dem Histogramm–Task mit der Prozedur `Start` in Modul HBR_B_IF eine Liste der nun zu erstellenden Histogramme, die aus den Science–Functions abgeleitet sind. Daraufhin können Histogramme integriert werden, siehe auch Kapitel 5.5. Außerdem wird das Lesen im Histogramm–Lesetask wieder aufgenommen, sofern es zuvor angehalten worden war (Prozedur `Resume`).

Bearbeiten von Einzelereignissen Bevor Einzelereignisse bearbeitet werden können, muss über die Schnittstelle `Buffer_Available` des Moduls HBR_A_IF, das eine Verbindung zum HBR-A–Task ist, abgefragt werden, ob Daten zur Verfügung stehen. Wenn dem so ist, können diese aus einem Puffer des Moduls HBR_A_IF ausgelesen werden. Der in dem Puffer gültige Bereich muss dabei mit der Funktion `Get_Buffer_Range` bestimmt werden. Die Einzelereignisse werden nacheinander untersucht und je nach Typ unterschiedlichen Bearbeitungsroutinen (in den Modulen S1 bis S4 und S8) übergeben, sofern der Telemetriepuffer für diesen Typ nicht über der Schwelle liegt. Innerhalb dieser

[11] Die andere Möglichkeit wäre gewesen, an VETO ein Relais–Pulse zu schicken, das den gleichen Effekt hat. Da aber sicher sein muss, dass VETO **ab**geschalten wird und nicht etwa aus Versehen (ein Puls wechselt nur den Zustand) angeschalten wird, wurde der anderen Methode der Vorzug gegeben.

Prozeduren findet die Kompression und Verpackung statt, woraufhin die gebildeten Telemetriepakete abgeschickt werden. Siehe auch [27]. Zusätzlich werden die Ereignisse typsortiert gezählt und eine Gesamtrate gebildet. Nicht erkannte Ereignisse werden ebenfalls gezählt, aber nicht weiterbearbeitet. Ist ein Puffer vollständig gelesen und verarbeitet, wird das Einlesen wiederholt.

Umschalten zwischen Point/Slew Nach dem Abarbeiten eines Puffers prüft die IASW, in welchem Status der Satellit (Pointing oder Slew) ist. Hat sich dieser geändert, so muss die HEPI umkonfiguriert werden. Dazu wird zunächst der Zugriff auf die Low speed Serial Line angefordert, aber nicht gewartet (Prozedur `announce_lsl` des Moduls LINE_MANAGER), sondern bis zum Freiwerden der Low speed Serial Line weiter ausgelesen.[12] Ist der Zugriff gewährt, so wird das HEPI–Status–Register analog zur Initialisierung geändert.

Resynchronisation Wenn die Datenrate zu gering ist oder zu viele unerkannte Ereignisse ausgelesen wurden, muss die HEPI neu initialisiert werden. Dazu bietet das Modul eine Prozedur namens `Resync_HEPI` an, die nach einer Pufferbearbeitung eine Synchronisation analog der beim Initialisieren durchgeführten Methode anstößt.

Beenden des Science–Mode Wenn das Modul MODE_DISPATCHER, ausgelöst durch ein Telekommando, die Prozedur `Stop` aufruft, so wird die Schleife der Ereignisbearbeitung abgebrochen. Daraufhin werden von allen Bearbeitungsroutinen die noch offenen Telemetriepakete geschlossen und abgeschickt. Schließlich wird noch an den Histogramm–Task und den Histogramm–Lesetask ein Signal geschickt, welches ein Beenden der Integration bzw. des Auslesens bewirken. Generell wird bei Start, Resynchronisation und Beenden des Science–Mode ein Science–Housekeepingpaket vom Modul SCIENCE_HK abgeschickt wird, in dem unter anderem auch die Ereigniszähler und die Synchronisationszeit stehen.

5.2.3 Diagnostic-Mode

Der **Diagnostic–Mode** ist analog dem Science–Mode aufgebaut. Der wesentliche Unterschied ist, dass keine Initialisierung der HEPI stattfindet und keine Histogramme gebildet werden. Es können drei Telemetrietypen erzeugt werden: Diagnostic–Daten von ISGRI und PICsIT, und ein Telemetriepaket, in das die gelesenen Einzelereignisse unverändert geschrieben werden. Alle diese Typen können, anders als im Science–Mode, über globale Variablen an–oder abgeschaltet werden.[13] Diese Bearbeitung der eingelesenen Ereignisse findet in dem Modul DIAGNOSTIC bzw. TRANSPARENT statt [27].

[12] Siehe auch Kapitel 4.2.1.

[13] Die Generierung von Diagnostic–Telemetriepakete kann über die Telekommandos TC(9,4) und TC(9,5) an– bzw. abgeschalten werden.

5.3 Der Telekommand-Task

5.3.1 Setzen/Lesen von Task–Parametern

TID	FID	Beschreibung
16	0	Schwellwert: CdTe–Energie$_{min}$
16	1	Schwellwert: CdTe–Energie$_{max}$
16	2	Schwellwert: Compton–Energie$_{max}$
16	3	Schwellwert: Noisy Pixel Rate$_{m}in$
17	0	Automatismus: Point/Slew an/aus
17	1	Automatismus: Reaktion auf Ekliptik–an/aus
17	2	Automatismus: Reaktion auf Strahlungsgürtel an/aus
17	3	Automatismus: Noisy Pixel Behandlung an/aus
17	4	Automatismus: Notabschaltung an/aus
17	5	Automatismus: HEPI–Resynchronisation an/aus
17	6	Automatismus: Reaktion auf hohe Strahlungs an/aus
18	0	Integrationszeit für Science–Histogramme
18	1	Integrationszeit für Kalibrations–Histogramme
32	0	Tabelle: Energiekorrektur–LUT
32	1	Tabelle: Anstiegszeit–LUT
32	2	Tabelle: Telemetrie–Schwellwerte
32	3	Tabelle: Strahlungsschwellwerte
32	16	Tabelle: Science–Funktionen Standard
32	17	Tabelle: Science–Funktionen PPM
32	18	Tabelle: Science–Funktionen Polarimetrie
128	0	Setze Electrical Configuration Register (ECR)
144	X	Kontexttabelle Wiederherstellen: X bezeichnet die Subinstrument–Adresse
145	X	Kontexttabelle Sichern: X bezeichnet die Subinstrument–Adresse
146	X	Lade Kontexttabelle: X bezeichnet die Subinstrument–Adresse
192	X	Sende Externes Telekommando an HEPI; X ist der Kommando–Identifier (CID)
193	X	Sende Externes Telekommando an ISGRI; X ist der Kommando–Identifier (CID)
194	X	Sende Externes Telekommando an PICsIT; X ist der Kommando–Identifier (CID)
195	X	Sende Externes Telekommando an VETO; X ist der Kommando–Identifier (CID)
208	0	Setze HEPI–Register Typ II in der IASW

Tabelle 5.3: Gültige Task–Parameter beim Laden

Telekommandos werden von der IASW in einem eigenen **Telekommand–Task** verarbeitet, dessen Eintrittspunkt die Prozedur `Start` im Modul TC_DECODER ist. Der Task ruft nach dem Start die CSSW–Prozedur `Get_TC` auf und wartet, bis ein Telekommando eintrifft. Wenn ein neues Telekommando angekommen ist, wird zunächst der Typ und der Subtyp untersucht und eine entsprechende Bearbeitungsprozedur aufruft. Die Telekommandos »Puls–Kommando« TC(2,1), und die Telemetrie–Status–Telekommands TC(9,1), TC(9,4) und TC(9,5) werden direkt im Modul TC_DECODER verarbeitet. Die Telekommandos »Start–Task« TC(5,1), »Stoppe–Task« TC(5,2) rufen eine Prozedur des Histogramm–Lesetasks auf, sofern die Task–ID und Function–ID stimmen.

Die Verwaltung von Parametern, Tabellen und externen Telekommandos, findet im Modul TASK_PARAMETER statt. Die Schnittstelle besteht aus zwei Prozeduren: `Load_Task_Parameter` und `Report_Task_Parameter`, denen unter anderem die TID/FID übergeben werden. Mit diesen Prozeduren können die Parameter der IASW gesetzt und gelesen werden. In Tabelle 5.3 sind die Telekommandos zum Setzen von Parametern aufgelistet, während Tabelle 5.4 alle Telekommandos zum Parameter–Lesen enthält.

TID	FID	Beschreibung
16	0	Schwellwert: CdTe–Energie$_{min}$
16	1	Schwellwert: CdTe–Energie$_{max}$
16	2	Schwellwert: Compton–Energie$_{max}$
16	3	Schwellwert:Noisy Pixel Rate$_{m}in$
18	0	Integrationszeit für Science–Histogramme
18	1	Integrationszeit für Kalibrations–Histogramme
32	0	Lese Tabelle: Energiekorrektur–LUT
32	1	Lese Tabelle: Anstiegszeit–LUT
32	2	Lese Tabelle: Telemetrie–Schwellwerte
32	3	Lese Tabelle: Strahlungsschwellwerte
32	16	Lese Tabelle: Science–Funktionen Standard
32	17	Lese Tabelle: Science–Funktionen PPM
32	18	Lese Tabelle: Science–Funktionen Polarimetrie
64	0	Letztes Telekommando
65	0	Erzeuge Science–Housekeeping–Paket
146	X	Lese Kontexttabelle: X bezeichnet die Subinstrument–Adresse
147	X	Lese Prüfsumme einer Kontexttabelle: X bezeichnet die Subinstrument–Adresse
208	0	Lese HEPI–Register Typ II aus der IASW

Tabelle 5.4: Gültige Task–Parameter zum Lesen

Einzelparameterverwaltung Alle Einzelparameter, die in Tabelle 5.3 unter der Task–ID 16 bis 18 aufgeführt sind, sind in verschiedenen Modulen als globale Variablen deklariert. Diese Parameter sind in Tabelle 5.5 aufgeführt.

Modul	TID	FID	Beschreibung
S1	16	0	Schwellwert: CdTe–Energie$_{min}$
S1	16	1	Schwellwert: CdTe–Energie$_{max}$
S3	16	2	Schwellwert: Compton–Energie$_{max}$
handle_pixel_noisy	16	3	Schwellwert:Noisy Pixel Rate$_{m}in$
bcpk_handler	17	0	Automatismus: Point/Slew
bcpk_handler	17	1	Automatismus: Reaktion auf Ekliptik
bcpk_handler	17	2	Automatismus: Reaktion auf Strahlungsgürtel
hk	17	3	Automatismus: Noisy Pixel Behandlung
bcpk_handler	17	4	Automatismus: Notabschaltung
science_mode	17	5	Automatismus: HEPI–Resynchronisation
bcpk_handler	17	6	Automatismus: Reaktion auf hohe Strahlungs
histogram_config	18	0	Integrationszeit für Science–Histogramme
histogram_config	18	1	Integrationszeit für Kalibrations–Histogramme
ecr	128	0	Electrical Configuration Register (ECR)
hepi	208	0	HEPI–Register Typ II in der IASW

Tabelle 5.5: Einzelparameter und ihre Module

Wenn ein Telekommando zum Lesen eines Parameters empfangen wurde, wird ein Telemetriepaket TM(5,4) gebildet und an das Modul TM_INTERFACE weitergegeben. Es wird nicht auf den Telemetriepuffer der CSSW geachtet (Ausnahme: Beim Auslesen von Parametertabellen), so dass unter Umständen diese Pakete verloren gehen können.[14] In Tabelle 5.6 ist die Struktur eines solchen Telemetriepaketes dargestellt.

Externe Telekommandos Vor dem Weiterleiten von externen Telekommandos an ein Subinstrument wird zunächst geprüft, ob eine Antwort erwartet wird. Dann wird direkt auf das Modul DETECTOR_INTERFACE zugegriffen und entweder die Prozedur `Send_CMD` oder `Read` (je nachdem, ob eine Antwort erwartet wird) aufgerufen, wobei zuvor die Low speed Serial Line blockiert wird. Die eventuell vorhandene Rückantwort wird in ein Telemetriepaket »Report Parameter« TM (5,4) gepackt und abgeschickt. Die Struktur dieses Telemetriepakets ist in Tabelle 5.7 dargestellt.

Tabellen-Verwaltung Alle Tabellen werden von einem Modul namens TABLE_HANDLER verwaltet, d.h. sie werden entweder dorthin geschrieben oder von

[14] Der Verlust eines Telemetriepaket wird mit einer Ereignismeldung quittiert, so dass in diesem Fall die Operation wiederholt werden muss. Der Fall sollte allerdings nicht eintreten, wenn die Schwellwerte der Telemetrie richtig gesetzt sind.

Word 1-3: Telemetriekopf

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Typ (5)				SubTyp (4)				0							

64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
Zeit (in sec)															

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
TID								FID							
Folgende Worte: Parameter															

Tabelle 5.6: Struktur eines Parameter–Telemetriepaket

Word 1-3: Telemetriekopf
Word 4-6: Typ/Subtyp, Zeit,TID/FID

96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
Länge (in Word)								CID							
Folgende Worte: Parameter															

Tabelle 5.7: Struktur der Rückantwort eines externen Telekommandos

dort gelesen. Wie man an den Telekommandos in Tabelle 5.3 und 5.4 sehen kann, haben interne Parametertabellen eine fortlaufende Tabellennummer, während für die externen Kontexttabellen die Subinstrument–Adresse als Tabellennummer gilt. In Tabelle 5.8 sind die Parametertabellen und Kontexttabellen mit ihren Modulen zusammengefasst. Wenn ein Tabellenblock mit einem Telekommando gesetzt werden soll, so wird dem TABLE_HANDLER der Block übergeben und dann von ihm ausgelesen und als Telemetriepaket abgeschickt. Dieses Telemetriepaket vom Typ TM(5,4) enthält als Parameter bei Kontexttabellen die Blocklänge, sowie die Blocknummer und den Tabellenblock. Wenn eine vollständige Tabelle ausgelesen werden soll, so wird alle 125 msec (1 RTC) ein Telemetriepaket abgeschickt, das bis zu drei Parameterblöcke enthalten kann.[15] Schlägt das Abschicken fehl, so wird gewartet. Überschreitet die kumulative Wartezeit eine Minute, bricht der Auslesevorgang ab und sendet eine Fehlermeldung.

Kontext–Übertragung In Tabelle 5.3 gibt es Telekommandos (TID =144,145), die eine Kontexttabelle von der IASW in ein Subinstrument

[15] Die Zahl der Blöcke in einem Telemetriepaket kann bestimmt werden, indem für jeden möglichen Block in einem Telemetriepaket geprüft wird, ob die Blocknummer ungleich Null ist. Die Blocknummer Null bezeichnet einen *leeren* Block.

Modul	TID	FID	Beschreibung der Tabelle
e_corrX	32	0	Energiekorrektur–LUT; X: Tabellenteil (1-5)
energy_correction	32	1	Anstiegszeit–LUT
tm_interface	32	2	Telemetrie–Schwellwerte
bcpk_handler	32	3	Strahlungsschwellwerte
science_functions	32	16	Science–Funktionen Standard
science_functions	32	17	Science–Funktionen PPM
science_functions	32	18	Science–Funktionen Polarimetrie
hepi_context_table	146	0	Kontexttabelle HEPI
picsit_context_table	146	64	Kontexttabelle PICsIT
isgri_context_table	146	X	Kontexttabelle ISGRI; X bezeichnet die Subinstrument–Adresse des ISGRI–MCE
veto_context_table	146	128	Kontexttabelle VETO

Tabelle 5.8: Tabellen und ihre Module

oder umgekehrt vom Subinstrument in die IASW übertragen.[16] Zum Sichern eines Kontexts wird die Prozedur `Download_Context`, zum Wiederherstellen die Prozedur `Upload_Context`[17] des entsprechenden Moduls aufgerufen. In ihnen wird Block für Block gelesen oder an das Subinstrument geschickt. Im Fehlerfalle wird abgebrochen. Nach der Blockübertragung fordert die IASW eine Prüfsumme von dem Subinstrument an und vergleicht sie mit der des eigenen Kontexts. Die schon erwähnten Kontext–Flags werden je nach Erfolg gesetzt. Das Telekommando ist allerdings (anders als für die Automatismen, siehe Kapitel 5.2.1) von ihnen unabhängig.

Prüfsumme Der Algorithmus zur Prüfsummenberechnung beschränkt sich auf das Aufsummieren der Worte eines Kontextes, um dann das Ergebnis mit Einer–Komplement zu negieren. Als Formel:

$$S = \mathbf{not} \sum_{i=1}^{N} \text{Kontext}[i]$$

mit S als Ergebnis und Kontext als Feld von Worten. Das Ergebnis ist auf ein Word von 16 Bit eingeschränkt. Die Berechnung der Prüfsumme in der IASW findet im Modul COMPUTE_CHECKSUM statt. Aus Geschwindigkeitsgründen ist sie in der Assemblersprache des 1750A–Prozessors programmiert.

Voriges Telekommando Zur Verifikation kann das letzte empfangene Telekommando abgefragt werden (siehe Tabelle 5.4).

[16] Der genaue Leser wird vielleicht fragen, wieso dafür nicht ein Telekommando vom Typ »Starte Task« (TC (5,1)) verwendet wurde, da dies hier doch am meisten angepasst wäre. So war es ursprünglich implementiert. Es hat sich jedoch herausgestellt, dass solche Telekommandos keine Function–ID haben, die aber zur Tabellenunterscheidung notwendig ist. Vgl. [22].

[17] Diese Begriffe sollen den Vorgang aus Sicht der IASW wiederspiegeln.

Dafür wird jedes erhaltene Telekommando, das kein Broadcast–Paket ist, zwischengespeichert. Bei Bedarf kann es angefordert werden, wobei einige Umformatierungen des Typs, wie er von der CSSW geliefert wird, notwendig sind. Insbesondere wird immer angenommen, dass das Telekommando keine Prüfsumme (CRC) hatte, da diese Information für die IASW nicht sichtbar ist.

5.3.2 Modus-Wechsel

Das Telekommando für den Moduswechsel (TC (5,5)) ruft in dem schon beschriebenen Modul MODE_DISPATCHER die Prozedur `Mode_Transition` auf. Diese prüft zunächst, ob der Moduswechsel erlaubt ist, und startet den nächsten bzw. stoppt den aktuellen Mode, je nachdem, ob der momentane Mode der Standby–Mode oder ein anderer war.

»Null–Übergänge« (momentaner und neuer Mode sind identisch) sind erlaubt, denn es gibt noch einen zusätzlichen Parameter, der den Point–Slew –Status bei ausgeschaltetem Point–Slew–Automatismus bestimmt. Dieser wird geprüft und bewirkt folgendes:

0	Keine Änderung
1	Setzte in Modul BCPK_HANDLER via Prozedur `Set_Pointing` den Status auf Pointing
2	Setzte via Prozedur `Set_Pointing` den Status auf Slew

Damit ist es möglich, auch bei ausgeschaltetem Point–Slew–Automatismus die verschiedenen Konfigurationen des Science–Modes zu verwenden.

5.3.3 Broadcast-Pakete

Die CDMU schickt alle acht Sekunden ein Telekommando an die IASW, welches allgemeine Informationen enthält. Wenn die IASW dieses Broadcast–Paket empfangen hat, ruft sie die Prozedur `Set_BCPK` des Moduls BCPK_HANDLER auf und übergibt ihr das Broadcast–Telekommando. Alle für IBIS relevanten Felder, die in Tabelle 3.6 aufgeführt sind, wird ausgewertet und gespeichert. Diese extrahierte Information ist von der IASW über verschiedene Funktionen zugänglich:

`Pointing:`

Gibt zurück, ob

1. Der Satellit im Standard–Betrieb (ACC–Pointing–Mode) und
2. Der Satellit gerade im Pointing–Mode ist und
3. Die Pointing–ID nicht null ist.

Dies gilt, sofern der Point–Slew–Automatismus an ist. Andernfalls wird der zuletzt mit TC (5,5) gesetzte Status zurückgegeben.

`On_Target`:
Gibt das ON–Target–Flag des letzten Broadcast–Pakets zurück, also ob der Satellit gerade stabil auf ein Objekt blickt. Zudem sollte er im Pointing–Mode sein.

`Hand_Over`:
Gibt zurück, ob der Satellit gerade die Empfangsstation wechselt.

`Goto_Standby`:
Gibt zurück, ob die IASW in den Standby–Mode gehen muss. Dies ist in folgenden Fällen nötig:

1. Der Satellit ist in der Ekliptik und der Ekliptik–Automatismus ist an.
2. Der Satellit ist im Strahlungsgürtel und der Strahlungsgürtel–Automatismus ist an.
3. Ein Not–Ausschalten (Emergency Save Acquisition Mode) bei angeschaltetem Notausschaltungs–Automatismus ist erforderlich.
4. Ein Ausschalten von der Bodenstation (Imminent Switch off) bei angeschaltetem Notausschaltungs–Automatismus ist erforderlich.
5. Einer der drei Strahlungsraten liegt über dem entsprechenden Schwellwert und der Strahlungs–Automatismus ist an.

`Save_Context`:
Gibt zurück, ob der Instrument–Kontext gesichert werden muss. Dazu muss gelten:

1. Der Satellit ist in der Eklipse und der Eklipse–Automatismus ist an.
2. Ein Ausschalten über die Bodenstation (Imminent Switch off) bei angeschaltetem Notausschaltungs–Automatismus ist erforderlich.

Der Satellit befindet sich für die IASW in der Ekliptik wenn die aktuelle Zeit nach der Ekliptik–Eintrittszeit und vor ihrer Austrittszeit liegt. Dies wird alle acht Sekunden überprüft.

`Veto_Off`:
Gibt zurück, ob das VETO–Subinstrument die Hochspannung des Photomultiplier ausschalten muss. Dies ist der Fall, wenn:

1. Der Satellit ist im Strahlungsgürtel und der Strahlungsgürtel–Automatismus ist an.
2. Ein Not–Ausschalten (Emergency Save Acquisition Mode) bei angeschaltetem Notausschaltungs–Automatismus ist erforderlich.

3. Ein Ausschalten von der Bodenstation (Imminent Switch off) bei angeschaltetem Notausschaltungs–Automatismus ist erforderlich.
4. Einer der drei Strahlungsraten liegt über dem entsprechenden Schwellwert und der Strahlungs–Automatismus ist an.

Analog zum vorigen Punkt befindet sich der Satellit im Strahlungsgürtel, wenn die aktuelle Zeit nach der Eintrittszeit und vor der Austrittszeit des Strahlungsgürtels ist.

`Slew_Time`:
: Gibt zurück, zu welcher Zeit der Satellit in den Slew–Mode gehen wird. Diese Zeit ergibt sich aus dem Anfang des Pointing (der durch den Wechsel in dem Broadcast–Paket von Slew in Pointing erkannt wird, siehe oben) und der Dauer des Pointing. Vorraussetzung ist, dass der Wechsel vom Slew zum Pointing zuverlässig erkannt wird, was nur der Fall ist, wenn alle acht Sekunden ein Broadcast–Paket geschickt wird. Wenn der Point–Slew–Automatismus aus ist, darf diese Funktion nicht verwendet werden.

`Get_Last_Pointing_Start`:
: Gibt die Startzeit des aktuellen Pointing zurück, der sich aus dem Wechsel von Slew nach Pointing definiert.

`Get_Last_Slew_Start`:
: Gibt analog die Startzeit des Slew zurück.

`Pointing_Id`:
: Gibt den letzte Pointing–Identifier zurück.

Diese Funktionen werden in der IASW teilweise zur Steuerung (Kontext sichern/wiederherstellen, VETO ausschalten, Beendigung des Science–Mode, Histogramme), teilweise für allgemeine Statusinformationen (Science–Housekeeping–Paket) verwendet.

5.3.4 Ungültige Telekommandos

Telekommandos müssen auf Konsistenz geprüft werden. Nicht immer werden Telekommandos akzeptiert, und die Ausführung kann unter Umständen fehlschlagen. Diese Fälle werden immer mit einer Ereignismeldung quittiert.

Ungültiger Modus Nicht in jedem Modus der IASW sind alle Telekommandos erlaubt. Dies betrifft den Science–Mode, den man über Telekommando starten und beenden darf, der aber keine sonstigen Telekommandos akzeptiert. In den anderen Modi (Standby, Diagnostik) sind alle Telekommandos erlaubt. Die Broadcast–Pakete sind in allen Modi erlaubt. Die Implementierung der erlaubten Telekommandos ist mit einer Matrix geschehen, die in jedem Modus und für alle Typ/Subtyp–Kombinationen die Akzeptanz eines Telekommandos bestimmt.[18]

[18] Mit diese Gültigkeitstabelle sind spätere Änderungen leicht implementierbar.

Ungültige Kategorien Wenn ein Telekommando eine nicht erkannte Typ/Subtyp–Kombination oder ein nicht implementiertes TID/FID–Paar enthält, wird es nicht ausgeführt. Stattdessen wird eine Ereignismeldung abgeschickt.

Ungültige Länge Jeder Parameter hat eine bestimmte Länge, die beim Setzen eingehalten werden muss:

Schwellwerte	16 Bit
Integrationszeiten	48 Bit
Parametertabellenblöcke	64+1Word
HEPI Register Typ II	11 Word

Fehlschlagen der Ausführung Ein Teil der Telekommandos beinhaltet komplexe Aufgaben. Dazu gehören Kommandos an die Subinstrumente, die Kontexttabellen–Übertragung, die Puls–Kommandos und das Auslesen von Tabellen. Wenn die entsprechenden Schnittstellen einen Fehler melden, wird die Ausführung des Telekommandos abgebrochen und eine Ereignismeldung abgeschickt.

Zähler Für jedes erfolgreich ausgeführte Telekommando und für jedes Broadcast–Paket erhöht sich ein Zähler für gültige Telekommandos, der im Housekeeping–Paket aufgeführt ist. Schlägt dagegen das Telekommando fehl, so erhöht sich ein Zähler für ungültige Telekommandos.

5.4 Der Housekeeping-Task

Der **HK–Task** zum Auslesen und Abschicken von Housekeeping–Paketen verwendet im wesentlichen das Modul HK, dessen Eintrittspunkt die Prozedur `Start` ist. Diese Prozedur wartet auf das Hardwaresignal BCP 1 und beginnt vier Sekunden nach dessen Empfang mit dem Auslesen der Housekeeping–Daten. Das heißt, dass vom Acht–Sekunden–Zyklus vier Sekunden zum Auslesen von Housekeeping–Daten vorgesehen sind.

Einsammeln von Housekeeping–Daten Beim Auslesen sammelt der Task zunächst alle relevanten Daten der IASW und packt sie in das Housekeeping–Feld der IASW. Die Daten sind in Tabelle 5.9 zusammengestellt. Zusätzlich werden Analogwerte von PICsIT und VETO gelesen.[19]

Danach können von den Subinstrumenten die Housekeeping–Daten eingesammelt werden, sofern das entsprechende Instrument angeschalten ist. Dies kann mit dem Electrical Configuration Register (ECR) festgestellt werden. Zum Auslesen wird von den Schnittstellenmodulen der Subinstrumente das entsprechende Low speed Serial Line–Kommando mit der in diesen Modulen definierten Housekeeping–Struktur verwendet.

[19] Analogwerte müssen möglichst frühzeitig ausgelesen werden, da sonst die Konvertierung mit der Housekeeping–Erzeugung der CSSW kollidieren kann.

Jede ausgelesene Housekeepingstruktur kann Ereignismeldungen enthalten; siehe auch [33]. Diese muss der HK–Task separieren und einzeln über das Modul OEM_INTERFACE der CSSW übergeben. Die eigentlichen Housekeeping–Daten werden als Block in die Housekeeping–Struktur der IASW kopiert. Zuletzt muss diese Housekeeping–Struktur abgeschickt werden. Dies geschieht mit der CSSW–Prozedur `Put_IASW_HK_TM`.

Byte	Bit	Beschreibung	aus Modul
0	0	Electronic Configuration Register	ECR
2	0	PICsIT 5V (analog)	ANALOG_IF
3	0	PICsIT Watchdog (bilevel)	ANALOG_IF
3	1	VETO Watchdog (bilevel)	ANALOG_IF
4	0	VETO VECU1 (analog)	ANALOG_IF
5	0	VETO VECU2 (analog)	ANALOG_IF
6	0	VETO Photomultiplier (analog)	ANALOG_IF
11	0	Aktueller Mode	MODE_DISPATCHER
11	7	Point Slew status	BCPK_HANDLER
12	0	Funktionen im Diagnostic Mode	DIAGNOSTIC_MODE
13	0	Funktionen im Science Mode	SCIENCE_MODE
15	0	Context Save Flags	INSTRUMENT_CONFIG
17	0	Context Restore Flags	INSTRUMENT_CONFIG
19	0	Point–Slew–Automatismus	BCPK_HANDLER
19	1	Eclipse–Automatismus	BCPK_HANDLER
19	2	Strahlungsgürtel–Automatismus	BCPK_HANDLER
19	3	Noisy_Pixel–Automatismus	HK
19	4	Notabschaltungs–Automatismus	BCPK_HANDLER
19	5	HEPI–Resynchronisations–Automatismus	SCIENCE_MODE
19	6	Strahlungsmessungs–Automatismus	BCPK_HANDLER
20	0	Abgeschickte Telemetriepakete	TM_INTERFACE
24	0	Verlorene Telemetriepakete	TM_INTERFACE
28	0	Status der Telemetrie (für jeden TM–Typ)	TM_INTERFACE
41	0	Gültige Einzelereignissrate	HK
43	0	Ungültige Einzelereignissrate	HK
45	0	Histogramm–Blöcke	READ_HISTOGRAM
47	0	Zähler von Ereignismeldungen (OEM)	OEM_INTERFACE
49	0	Interne Kommandos an HEPI	HEPI
50	0	Interne Kommandos an ISGRI	ISGRI
51	0	Interne Kommandos an PICsIT	PICSIT
52	0	Interne Kommandos an VETO	VETO
53	0	Letztes angesprochenes Subinstrument (Adresse)	DETECTOR_INTERFACE
54	0	Gültige Telekommandos	TC_DECODER
55	0	Ungültige Telekommandos	TC_DECODER

Tabelle 5.9: IASW–Housekeeping–Daten

Zusätzliche Housekeeping Für ISGRI, PICsIT und VETO müssen in Zeitabständen von 32 sec bis 30 min weitere Housekeeping–Daten ausgelesen und abgeschickt werden. Diese zusätzlichen Housekeeping–Pakete gehen als normale Telemetriepakete über das Modul TM_INTERFACE ab und verwenden nicht die CSSW–HK–Schnittstelle. Das Auslesen, Abschicken und die Zeitkontrolle geschieht mit der Prozedur `Get_Add_HK` aus dem Modul ADD_HK[20], der nur das Electrical Configuration Register (ECR) übergeben werden muss. Die Perioden der verschiedenen Housekeeping–Pakete sind:

Subinstrument	Periode (in sec)
ISGRI	32
PICsIT	32
VETO	1800

Überwachung der Ereignisrate Da von allen Tasks nur der Housekeeping–Task strikt zyklisch mit einer Periode von acht Sekunden läuft, muss er noch einige weitere Überwachungsaufgaben übernehmen. Im Science–Mode muss die Zahl der Einzelereignisse gelesen und zurückgesetzt werden (sie erscheint auch in den Housekeeping–Daten, siehe Tabelle 5.9). Wenn diese Rate gültiger Einzelereignisse zu gering oder die der ungültigen zu hoch ist, muss HEPI resynchronisiert werden, sofern der HEPI–Resynchronisations–Automatismus an ist. Dazu ruft der HK–Task die Prozedur `Resync_HEPI` des Moduls SCIENCE_MODE auf, die schon beim Main–Task beschrieben ist. Die Schwellen liegen derzeit fest kodiert bei minimal 1000 gültigen Ereignissen bzw. maximal Null ungültigen Ereignissen pro acht Sekunden.

Ausschalten von *Noisy Pixels* Ist im Science–Mode der Noisy–Pixel–Automatismus an, so werden die von PICsIT gelesenen Ereignismeldungen (OEM) untersucht. Meldet eine einen Überlauf der PICsIT–Ereignispuffer, so wird angenommen, dass dies durch ein defektes und rauschendes Pixel erzeugt wurde, das ausgeschalten werden muss. Die Schwierigkeit liegt darin, herauszufinden, welches das defekte Pixel ist. Dazu muss von der HEPI für jedes Pixel des entsprechenden PICsIT–Moduls der dort angelegte Zählstand der sogenannten **Ratemeter** ausgelesen werden. Wenn für ein bestimmtes Pixel die Zählrate über einer einstellbaren Schwelle liegt, wird über ein Kommando an PICsIT das Pixel ausgeschalten und eine entsprechende Ereignismeldung abgeschickt.

Wenn jedoch für mehr als ein Pixel die Schwellen überschritten sind, kann nicht mehr von einem rauschenden Pixel ausgegangen werden, sondern es muss von einer erhöhten Strahlung (etwa einem Sonnen–Ausbruch) ausgegangen werden. In diesem Falle darf kein Pixel ausgeschalten werden. Diese etwas verwickelte Prozedur wird im Modul HANDLE_PIXEL_NOISY abgearbeitet.[21]

Für ein korrektes Funktionieren dieses Ablaufs müssen in HEPI die Zählstände (Ratemeter) der einzelnen Pixel regelmäßig zurückgesetzt werden, da eine Rate und nicht die absolute Zahl der Ereignisse relevant ist. Deshalb müssen nach der Noisy–Pixel–Behandlung die Zählstände in der HEPI mit einem Kommando gelöscht werden. Außerdem sollten sich während der Noisy–Pixel–Behandlung die Zählstände nicht ändern. Deshalb wird vor der Noisy–

[20]Dieses Modul ist von der Firma LABEN, Mailand entwickelt worden.
[21]Auch dieses Modul ist von der Firma LABEN entwickelt.

Pixel–Behandlung die Zählfunktion in HEPI angehalten und hinterher wieder gestartet.

5.5 Die Histogramm-Tasks

Es hat sich für die Implementierung der Histogramm–Bearbeitung am sinnvollsten erwiesen, zwei Tasks zu verwenden: einen, der die komplette Integration von Histogrammen steuert, und einen, der das Auslesen, Verpacken und Abschicken aller Histogramme erledigt. Von beiden Tasks werden sowohl Science– wie auch Kalibrationshistogramme bearbeitet. Dies mag schwierig erscheinen, weil beide Histogrammtypen unterschiedliche Integrationszeiten haben können. Da aber die Integration relativ leicht aufteilbar ist und beim Auslesen immer nur ein Histogram–Block nach dem anderen von der HEPI gelesen werden kann, ist dies der günstigste Ansatz.

5.5.1 Histogramm–Integrations–Task

Die **Integration von Histogrammen** wird in einem Task vorgenommen, der **Histogramm–Task** genannt wird. Sein Eintrittspunkt ist die Prozedur `Initialise` im Modul HBR_B_IF.[22] Nach der Initialisierung ist der Task zunächst suspendiert, d.h. gestoppt. Erst wenn der Main–Task die Prozedur `Start` aufruft, wird er lauffähig gemacht. Daraufhin prüft der Task nach den in Kapitel 3.4.3 beschriebenen Kriterien, ob ein Histogramm (Science– oder Kalibrationshistogramm) integriert, angehalten oder gestoppt werden muss. Für die notwendigen Befehle an HEPI wird dabei auf das Modul HISTOGRAM_OP zurückgegriffen, in dem alle wichtigen Operationen für die Histogrammbearbeitung definiert sind (Starten, Anhalten und Löschen bestimmter Histogramme). In diesem Modul ist auch ein Variablen–Typ `T_Histogram` definiert, der alle relevanten Informationen eines Histogramms (Startzeit, Stoppzeit, belegter Speicher etc.) enthält.

Wenn die IASW den Science–Mode verläßt, so muss die Integration gestoppt werden. Dazu schickt die von dem Main–Task aufgerufenen Prozedur `Stop` ein Signal an den Histogramm-Task, der die Integration anhält und sich selbst suspendiert.[23] Wenn eine neue Histogramm–Integration gestartet wird und bereits ein Histogramm integriert worden ist, dann wird dem Histogramm–Lesetask ein Start–Signal mit der Information über dieses Histogramm geschickt. Außerdem wird ihm die beabsichtigte Integrationsdauer mitgeteilt.[24]

[22]Der Modulname „HBR–B–Interface“ ist missverständlich. Er rührt daher, dass ursprünglich die gesamte Histogrammverwaltung in einem Task durchgeführt werden sollte, der eine Schnittstelle zur High speed Serial Line B (HBR B) sein sollte. Dieser Task ist aber dann aufgeteilt worden.

[23]Dieses Vorgehen ist einem unkontrolliertem Anhalten eines Tasks von außen vorzuziehen.

[24]Dies sollte nur geschehen, wenn kein Histogramm mehr ausgelesen wird. Der Histogramm–Lesetask läuft aber sicher nicht mehr: Angenommen, Puffer A wird noch ausgelesen, ist aber nicht leer, Puffer B wurde bereits integriert und soll ausgelesen werden, d.h. kein Puffer ist frei für eine Integration $\Longrightarrow$ Widerspruch!, da laut obiger Vorraussetzung das Start–Signal bei Beginn einer Integration geschickt wird. $\square$

Der Task ist zyklisch mit einer Periode von 125 Millisekunden (einem RTC). Allerdings kann es Schwierigkeiten geben, wenn auf die Low speed Serial Line zugegriffen und gewartet werden muss, da dann die Task–Periode überschritten werden kann. Das Problem kann gelöst werden, indem der Task für die Zeit, in der er auf die Low speed Serial Line zugreift, nicht–zyklisch gemacht wird. Siehe auch Kapitel 5.8.2.

5.5.2 Histogramm–Lesetask

Der **Histogramm–Lesetask** wird im Modul READ_HISTOGRAM in der Prozedur `Initialise` gestartet. Er prüft zunächst nach, ob ein Histogramm ausgelesen werden muss. Wenn nicht, so wartet er auf ein Start–Signal des Histogramm–Tasks, das bei Beginn einer neuen Integration geschickt wird.

Wenn er jedoch gerade läuft, kann er durch ein Signal angehalten werden. In diesem Fall wartet er zwischen zwei Auslesevorgänge in der Prozedur `Wait`, die noch beschrieben wird, auf das nächste Startsignal. Dieser Task ist ebenfalls zyklisch mit einer Periode von 125 Millisekunden (1 RTC). Das Auslesen selber besteht aus folgenden Schritten:

- Histogrammblock bei der HEPI bestellen.
- Mindestens vier Millisekunden warten, bis die HEPI den Block im FIFO bereitgestellt hat.
- Durch Abfragen einer im Modul HEPI definierten Relais–Leitung prüfen, ob der Block auch tatsächlich abholbereit ist.
- Block auslesen.
- Block komprimieren und in mehreren Telemetriepaketen abschicken.

Das Anfordern und Auslesen geschieht dabei wieder mit Routinen aus dem Modul HISTOGRAM_OP. Damit dies möglichst effizient geschieht, wird in einem Task–Zyklus ein Histogramm–Block bei der HEPI angefordert und mit Beginn des nächsten Zyklus über die High speed Serial Line (HBR B) ausgelesen und verarbeitet. Danach kann der nächste Histogrammblock angefordert werden, und der Prozess geht von vorne los.

Mischen von Histogrammtypen Science–Histogramme und Kalibrationshistogramme müssen gleichzeitig ausgelesen werden. Da dies möglichst gut über den Integrationszeitraum des laufenden Histogramms verteilt sein muss, sollte nicht mit der eben beschriebenen Methode mit maximaler Geschwindigkeit ausgelesen werden. Ausserdem muss darauf geachtet werden, welcher Histogramm–Block angefordert worden ist, damit er von der richtigen Routine komprimiert und abgeschickt wird. Daher werden beide Histogrammtypen in separaten Prozeduren behandelt, die aber zwischen dem Anfordern und Auslesen eine spezielle Prozedur namens `Wait` aufrufen. In dieser wird auf den nächsten Zyklus gewartet. Damit aber nun das Auslesen möglichst gut verteilt wird, werden aus der bekannten Integrationszeit **Wartezyklen** bestimmt, die festlegen, wie viele Zyklen zwischen zwei Ausleseoperationen verstreichen müssen. Jeder Histogrammtyp bekommt außerdem einen Zähler, der anfangs auf der Zahl der Wartezyklen steht, aber mit jedem Aufruf der Prozedur `Wait` erniedrigt wird. Wenn für ein Histogrammtyp der Zähler kleiner Null ist, darf der Histogrammblock ausgelesen werden, andernfalls wird das Auslesen übersprungen. Mit dieser an das Polling erinnernden Verfahren ist ein gleichmäßiges Auslesen und Bearbeiten von Histogrammblöcken gewährleistet.

Details Bei jedem Auslesen wird die Leseadresse im Histogramm um 1024 Byte erhöht, was einem Histogrammblock entspricht. Wenn sie das Ende des Histogramms überschreitet, wird es als ≫leer≪ (empty) markiert. Dieser Zustand kann über eine Funktion abgefragt werden. Die Bearbeitung der Histogrammblöcke, die in den Modulen S5 (Kalibrations–Histogramm), S6 (Polarimetrie–Histogramm) und S7 (CsI–Histogramm) stattfindet, erfordert einen zusammenhängenden Datenstrom, da bei unstetigen Blocknummern ein neues Telemetriepaket angelegt werden muss. Daher wird bei nicht–kontinuierlicher Adresse, wenn etwa ein Histogrammblock verloren geht, eine Ereignismeldung abgeschickt und die Bearbeitungsroutine zurückgesetzt. Ist es wegen eines vollen Telemetriepuffers nicht möglich, ein Telemetriepaket abzuschicken, dann wartet die entsprechende Bearbeitungsroutine maximal 10 RTC (1.25 sec).

Fehler Da die Packroutinen so sensibel auf die Adressen achten, kann kein Auslesefehler einfach übergangen werden:

- Schlägt die Anforderung fehl, so wird die Operation nach einer Zeit von 125 msec wiederholt.
- Wenn ein angeforderter Histogrammblock nicht im nächsten Zyklus verfügbar ist, wird eine Zeit von 5×125 Millisekunden gewartet und periodisch nachgefragt. Ist dennoch kein Block verfügbar, wird die Operation mit einer Fehlermeldung abgebrochen, und die Adresse erhöht. Die Packroutine wird dann zurückgesetzt.

5.6 Die Schnittstellen-Tasks

In der IASW gibt es zwei weitere Tasks, die keine besonderen Datenverarbeitungsaufgaben haben, sondern auf bestimmte Hardwaresignale reagieren. Dies sind der HBR–A–Task, der das Einlesen der Einzelereignisse übernimmt, und der Line–Manager–Task, welcher die Umsetzung und Kontrolle von Signalen, die die seriellen Schnittstellen betreffen, verwaltet.

5.6.1 HBR A–Task

Einzelereignisse werden im Main–Task bearbeitet. Während dieser Bearbeitung muss die IASW neue Einzelereignisse einlesen. Das Einlesen geschieht asynchron, d.h. bestimmte Hardwaresignale müssen abgewartet werden. Gerade der

Main–Task sollte aber so wenig Zeit wie möglich mit Warten verbringen, da sonst die Leistung der Datenverarbeitung sinkt. Der **HBR–A–Task** entlastet den Main–Task von dieser Aufgabe, indem er das komplette Einlesen der Einzelereignisse von der HEPI überwacht.

Schnittstellen Der Eintrittspunkt des HBR–A–Task ist die Prozedur `Initialise` im Modul HBR_A_IF, in dem der gesamte Task implementiert ist. Die Funktion `Buffer_Available` muss vor dem Auslesen von Daten benutzt werden, um zu prüfen, ob Daten eingelesen wurden und zur Bearbeitung anstehen. Der Puffer, der die Daten von der HEPI enthält, ist global als Feld mit zwei Teilen implementiert. Ein Teil dient als Lesepuffer, aus dem die IASW die Einzelereignisse lesen darf. Der andere Teil kann mit neuen Daten beschrieben werden, darf folglich nicht zum Lesen verwendet werden. Sind neue Daten verfügbar, so kann man mit der Prozedur `Get_Buffer_Range` der neue Lesebereich des Puffers bestimmt werden.[25]

Interner Ablauf Die Datenübertragung besteht aus mehreren Schritten:

Zunächst sind beide Bereiche des Einlesepuffers leer, was mit einer internen Variable `Buffer_Filled` = `FALSE` festgehalten wird. Der Aufruf von `Buffer_Available` schickt ein Signal an den HBR–A–Task, der prüft, aus welchem Pufferteil gerade gelesen wird. Angenommen, der Lesepuffer sei bei Beginn auf Puffer 1 gesetzt. Da der andere Pufferteil (2) leer ist, startet der Task eine Übertragung via High speed Serial Line in Puffer 2, sofern die HEPI das Vorhandensein von Daten (über einer Relais–Leitung, die im im Modul HEPI zugänglich ist) mitteilt.

War die Übertragung erfolgreich, so markiert der Task den Pufferteil 2 als »gefüllt«, und ein folgender Aufruf von `Buffer_Available` setzt den Lesepuffer auf Puffer 2 und gibt »TRUE« zurück. Gleichzeitig wird der leere Puffer 1 vom HBR–A-Task gefüllt. Das Auslesen des Puffers (2) im Main–Task kann beginnen. Der dafür erforderliche Prozeduraufruf von `Get_Buffer_Range` gibt den Bereich von Puffer 2 zurück und deklariert Puffer 2 als ausgelesen (`Buffer_Filled`(2) = `FALSE`). Dennoch wird Puffer 2 noch nicht überschrieben. Dies geschieht erst mit dem nächsten Aufruf von `Buffer_Available`, weil immer nur der »Nicht–Lese–Puffer« (momentan Puffer 1) gefüllt werden darf. Der nächste Lesezyklus aus Puffer 1 kann beginnen.

5.6.2 Line-Manager — Verwaltung der seriellen Leitungen

Die in Kapitel 4.1 angesprochenen Hardwaresignale betreffen größtenteils Informationen über den Status einer Übertragung via serieller Schnittstelle. In Tabelle 5.10 sind diese zusammengefasst. Da nun faktisch alle Tasks auf die seriellen Leitungen zugreifen, wäre es eine große Resourcenverschwendung, jeden

[25] Da das Pufferfeld global definiert ist, könnte man auch ohne Aufruf dieser Prozedur andere Einzelereignisse auslesen. Dies wird softwaremäßig nicht abgefangen, würde aber zu gravierenden Fehlern führen und ist daher nicht erlaubt. Genausowenig ist die Verwendung der Prozedur `Get_Buffer_Range` ohne Benutzung der Funktion `Buffer_Available` erlaubt.

Name (CSSW)	Beschreibung
`LSL_End_Of_Transmission`	Ende einer Übertragung der Low speed Serial Line (Hin oder zurück).
`HSL_End_Of_Message`	Ende eines Lesens von der High speed Serial Line.
`LSL_Parity_Error`	Paritätsfehler auf der Low speed Serial Line.
`LSL_Overrun_Error`	LSL–Register der DPE ist nicht rechtzeitig ausgelesen worden.
`HSL_Fifo_Full_Error`	FIFO–Überlauffehler auf der HSL der DPE. (Zu große Übertragungsgeschwindigkeit im Vergleich zum DMA).

Tabelle 5.10: Hardware–Signale der Datenübertragung

Task mit diesen Hardwaresignalen zu verknüpfen.[26] Deshalb ist ein zusätzlicher Task, nämlich der **Line–Manager–Task**, eingeführt worden, der die Hardwaresignale in Softwaresignale transformiert. Dazu wartet der Task auf alle Hardwaresignale, die mit den seriellen Schnittstellen zu tun haben, die in Tabelle 5.10 aufgeführt sind, und schickt die Information als Signal an den Task, der die entsprechende serielle Leitung blockiert, weiter. Diese Signale unterscheiden allerdings nur zwischen »Übertragung erfolgreich« und »Übertragung fehlgeschlagen«. Zur Fehleranalyse schickt der Line–Manager–Task beim Auftreten von Fehler–Signalen eine Ereignismeldung (Fehlermeldung auf unterster Ebene) an die Bodenstation, die die volle Information über die Art des Fehlers enthält.

Kontrolle der seriellen Schnittstelle Der Vorteil bei einer solchen Umsetzung ist eine Überwachung darüber, ob die serielle Schnittstelle korrekt blockiert worden ist. Denn wenn ein Hardwaresignal das Ende einer Übertragung meldet, obwohl die entsprechende serielle Leitung nicht blockiert wurde, so liegt ein intrinsischer Fehler der Software vor. In diesem Fall wird eine Ereignismeldung abgeschickt. Vergleiche auch Kapitel 4.2.1.

5.7 Das Zusammenspiel der Tasks

Bis jetzt sind alle Tasks einzeln mit ihrer Funktionalität vorgestellt worden. Die Frage ist nun, ob sie auch, wenn sie zusammen laufen, sich nicht gegenseitig beeinträchtigen. Für die Beantwortung dieser Frage müssen zyklische und nichtzyklische[27] Tasks (regelmäßige bzw. sporadisch arbeitende Tasks) auseinandergehalten werden und verschiedene Punkte überprüft werden:

1. Blockiert ein Task einen anderen länger, als es für die Funktionalität dieses Tasks erlaubt ist?

[26] Die Reaktionszeit auf diese Signale sinkt dadurch, weil die CSSW mehr Tasks bei dem Auftreten eines Hardware–Interrupts bedienen muss. Daher kann die Zahl der Tasks pro Hardwaresignal in der CSSW konfiguriert werden und sollte möglichst klein sein.

[27] Die Bezeichnung ist auch synchroner bzw. asynchroner Task.

2. Dauern zyklische Tasks länger als ihre definierte Zeitspanne?
3. Warten zyklische Tasks zu lange auf ein Signal?
4. Blockieren Tasks sich gegenseitig (dead lock)?

Wenn die Antwort in allen Punkten für alle Tasks negativ ist, kann von einem sicheren System ausgegangen werden.[28]

Task–Prioritäten Die wesentliche Gestaltungsmöglichkeit eines Multitaskingsystems ist die **Priorität**, mit der ein Task läuft. Sie wird in der IASW beim Initialisieren der Tasks vergeben und regeln, welcher nicht wartende und lauffähige Task (busy) als nächstes laufen darf. Die Prioritätsverteilung der IASW–Tasks mit CSSW–Tasks ist in Tabelle 5.11 festgehalten.[29]

5.7.1 Sicherheitsbetrachtungen

Line–Manager–Task Zunächst muss der Line–Manager–Task die höchste Priorität haben, weil er die empfangenen Hardwaresignale sofort an alle Tasks weitergeben muss (kein Task darf ihn wegen höherer Priorität blockieren). Dies ist unproblematisch, weil der Task sehr wenig Zeit bei der Umsetzung braucht (≤ 2 msec).

Housekeeping–Task Der Housekeeping–Task darf nicht von anderen Tasks blockiert werden, da er strikt alle acht Sekunden ein Housekeeping–Paket abschicken und verschiedene Überwachungen durchführen muss. Das bedeutet aber auch, dass er teilweise die Low speed Serial Line über längere Zeit blockiert. Daher dürfen sich die anderen Tasks nicht auf eine schnelle Verfügbarkeit der Low speed Serial Line verlassen (Außnahme: Der Main–Task). Damit ist Punkt 1 und 3 für den Housekeeping–Task abgedeckt. Punkt 2 ist durch eine großzügige Bearbeitungszeit von 4 Sekunden gewährleistet.

TC–Task Da die Steuerung ohne größere Blockierung erfolgen soll, ist der TC–Task mit relativ hoher Priorität ausgestattet. Dieser Task ist nicht zyklisch, braucht aber unter Umständen viel Rechenzeit und kann die Low speed

[28] In dem CSSW–Requirement–Dokument [21] wird vorgeschlagen, die Sicherheit des Multitaskingsystems dadurch zu garantieren, dass die nicht–zyklischen Tasks möglichst kurz laufen, und den zyklischen Tasks eine wohldefinierte Zeitspanne gegeben wird (verschiedene Margins). Meines Erachtens greift dieser Ansatz für die IASW von IBIS nicht vollständig, weil etwa gerade der nichtzyklische Main–Task einen großen Teil der Rechenzeit braucht. Er kann auch nicht einfach zyklisch gemacht werden, weil die Einzelereignisse von Natur aus statistisch auftreten und die Bearbeitungszeit sehr unterschiedlich ist.

Vielleicht wäre ein Design des Main–Tasks, das nur Einzelereignisse von HEPI ausliest und an verschiedene zyklische Tasks weiterleitet besser gewesen; es fragt sich aber, ob der Vorteil einer »geraderen« Struktur des Systems den Geschwindigkeitsnachteil infolge erhöhter Inter–Taskkommunikation wettgemacht hätte.

Anderseits muss nun die Untersuchung breiter ausfallen, als es bei dem vorgeschlagenen Vorgehen nötig gewesen wäre.

[29] Die absoluten Nummern können sich bei einer Feinkonfigurierung der IASW/CSSW noch ändern; die Prioritätsordnung sollte allerdings beibehalten werden.

Priorität	IASW–Task	CSSW–Task	Zyklisch
19		Astres Clock Task	
18		Minor Cycle	
17		DPE–State	
16		HW–Anomaly	
15	Line–Manager–Task		Nein
14	HK–Task		Ja
13	TC–Task		Nein
10	Histogram–Task		Ja[a]
9	Read–Histogram–Task		Ja[a]
7		TM–Cycle	
6		Current–TC	
5		TC–Input–Buffer	
4	HBR–A–Task		Nein
3	Main–Task		Nein
2		Memory Dump	
1		Idle Process	
0		Astres background task	

[a]Nicht streng zyklisch.

Tabelle 5.11: Task–Prioritäten

Serial Line auch länger blockieren. Dies kann aber nur im Standby–Mode oder Diagnostic–Mode der IASW geschehen; im Science–Mode ist nur das unkritische Telekommando »Mode ändern« erlaubt. In den anderen Modi spielt eine größere Belastung keine Rolle. Schlimmstenfalls sinkt im Diagnostic–Mode etwas die Bearbeitungsrate von Einzelereignissen. Histogramme sind im Diagnostic–Mode ohnehin nicht möglich. Deshalb kann davon ausgegangen werden, dass Punkt 1 berücksichtigt ist.

Histogramm–Tasks Aus ihrer Beschreibung ergibt sich, dass die Histogramm–Tasks zyklisch sind. Die Rechenzeit des Histogramm–Tasks ist gering, da nur einige Werte verglichen und die Low speed Serial Line bei einer Integration gestartet und gestoppt werden muss. Anders sieht es für den Histogramm–Lesetask aus, der die Komprimierung der Histogramme durchführt. Diese dauert etwa 65 Millisekunden, damit bleibt noch etwa 60 Millisekunden für die anderen Tasks, die aber ausreichend sind.

Es ist für diese Tasks kein Problem, wenn der Zugriff auf die Low speed Serial Line nicht sofort gewährt wird, da die Zeiträume, in denen sie Arbeiten (ein Histogramm alle 1000 Sekunden) relativ groß ist und eine Verzögerung kaum Folgen hat.[29] Die Tasks müssen allerdings darauf achten, dass sie nicht bei Periodenende laufen.[30] Auch bei hohem Rechenaufwand kann dies nicht

[29]Schlimmstenfalls wird etwas Meßzeit verschenkt, wenn ein Histogramm zu langsam ausgelesen wird. Dies kann man aber mit etwas Spielraum bei der Auslesegeschwindigkeit kompensieren.

[30]Die CSSW achtet bei zyklischen Tasks darauf, dass sie bei einem neuen Zyklus auf das

geschehen, es sei denn, ein Task wartet auf das Ende einer LSL–Übertragung, das jederzeit auftreten kann. Ein solcher Task Overrun wird in diesen Tasks dadurch behoben, dass sie sich während der Zeit einer Übertragung eines LSL–Kommandos aus der Tabelle zyklischer Tasks austragen, um sich danach wieder anzumelden.[31] Der Zugriff auf die Low speed Serial Line ist nur selten und mit etwa 10 Millisekunden[32] relativ kurz. Damit sind Punkt 1, 2 und 3 erfüllt.

HBR–A–Task Relativ unkritisch ist der HBR–A–Task, weil er zum einen nur auf die High speed Serial Line zugreifen muss, die er mit dem Histogramm–Lesetask teilt. Diese serielle Schnittstelle ist jedoch nach etwa fünf Millisekunden[33] wieder frei, so dass der Task nur unwesentlich warten muß. Auf der anderen Seite ist eine kleine Wartezeit unkritisch, da die Verarbeitung der Einzelereignisse wesentlich länger als das Einlesen dauert. Daher sind die ersten drei Punkte sicher erfüllt.

Main–Task Der IASW–Task mit der niedrigsten Priorität ist der Main–Task, weil er zum Einen einen Großteil der verfügbaren Rechenzeit braucht, auf der anderen Seite nicht zyklisch ist. Daher kann der Task frei mit der Rechenzeit umgehen, die für ihn bleibt. Diese nutzt er, indem der Task im Science–Mode soviele Einzelereignisse verarbeitet, wie es ihm möglich ist, ohne dass er auf irgendein Signal warten müsste.

Allerdings muss er im Science–Mode auch gelegentlich die Konfiguration der Funktionen ändern und dabei auf die Low speed Serial Line zugreifen. Dies geschieht in der schon beschriebenen Weise, so dass der Task nicht warten muss. Die Übertragung ist mit maximal 10 msec kurz, so dass die Resource »LSL « weitgehend den anderen Tasks zur Verfügung steht.

Im Standby–Mode des Main–Tasks gibt es keine Anforderungen, dass eine Aufgabe innerhalb kurzer Zeit zu erledigen ist (für die Kontextübertragung, die im Bereich von Minuten liegt, muss allerdings genügend Spielraum reserviert werden). Damit sind auch für diesen Task die ersten drei Punkte erfüllt.

Die CSSW–Tasks Betrachtet man die Tabelle 5.11, so finden sich dort auch Tasks der CSSW, die mitberücksichtigt werden müssen. Es muss allerdings nur die Resource »Rechenzeit« betrachtet werden, da die CSSW keine seriellen Leitungen verwendet, sondern sie nur zur Verfügung stellt. Die CSSW–Tasks mit den höchsten Prioritäten bereiten keine Schwierigkeiten, da sie nicht von der IASW beeinflußt werden.

Zeitsignal warten. Andernfalls wird eine Ereignismeldung »Task Overrun« erzeugt. Siehe Kapitel 5.8.2.

[31] Dies geschieht mit der CSSW–Prozedur `Delete_Period` bzw. `Define_Period`.

Dieses Vorgehen ist nicht ganz unproblematisch, wenn der Task sich kurz vor Periodenende als zyklisch anmeldet. Es gibt dafür noch keine definitive Lösung.

[32] Dies ergibt sich aus einer Übertragungsrate von 16KBaud, maxiaml 5 Word pro Kommando an die HEPI (Status–Registser setzen), und 11 Bit pro Byte, also $T = \frac{1}{16000/(8+3)} \times 5 \times 2 = 6.8$ msec ohne Spielraum.

[33] Der DMA–Interlace–Faktor (Speicherzugriffsfaktor) ist 32, die Geschwindigkeit etwa 13 MHz und ein Block enthält rund 1000 Word. Damit kommt man auf eine Zeit $T = \frac{1}{13 \times 10^6/1000/32} = 2.5$ Millisekunden. Der Rest ist für die Zugriffszeit auf die Routinen.

Bei den Tasks mit der Priorität 5,6 und 7 muss darauf geachtet werden, dass sie nicht ausgeschlossen werden. Dies ist allerdings auch bei dem hohen Rechenzeitverbrauch des Histogramm–Lesetask nicht der Fall. Von den CSSW–Tasks geringer Priorität gibt es noch zwei, die nicht vernachlässigt werden dürfen. Der Memory–Dump–Task wird zum Auslesen von Speicherwerten verwendet. Dies sollte allerdings nur im Standby–Mode der IASW geschehen, wenn diese nicht gar generell mit dem CSSW–Telekommando »Gehe zu CSSW–Started« angehalten wird.

Im Idle–Prozess der CSSW hingegen findet das sogenannte **RAM–Scrubbing** statt. Dabei prüft und korrigiert die CSSW periodisch den gesamten physikalischen Speicher der DPE. Für diesen Prozess gibt es keine spezifischen Anforderungen, wann er abzulaufen hat. Er darf nur nicht ganz ausgeschlossen werden. Daher ist der Main–Task so konzipiert, dass er zumindest im Standby–Mode jedes RTC (125 msec) wartet, um auch diesem Task Rechenzeit zu gewähren.

5.7.2 Dead–Lock–Analyse

Der letzte Punkt der Checkliste ist bei den Sicherheitsbetrachtungen offen geblieben. Es handelt sich bei **Dead Locks** um die Gefahr, dass zwei Tasks auf ein Signal je von dem anderen Task warten, etwa, weil jeder Task mehrere Resourcen benötigt, von denen jeder eine belegt hat.

Diese Situation ist für die IASW jedoch nicht gegeben. Denn es gibt zum einen keine Resourcen, welche weitere Resourcen erfordern, z.B. Anfordern der Low speed Serial Line und High speed Serial Line gleichzeitig. Das heißt, dass jeder Task eine Resource anfordern, warten, sie benutzen und schließlich wieder freigeben kann.

Können zum anderen Tasks sich gegenseitig blockieren, indem sie jeweils auf ein bestimmtes Signal (jetzt kein Hardwaresignal) eines anderen Tasks warten? Die IASW ist so angelegt, dass es immer ein klares hierarchisches Abhängigkeitsverhältnis (»Master–Slave«–Schema) existiert. So wartet weder der HK–Task noch der TC–Task auf Signale von anderen Tasks, sowenig wie der HBR–A–Task und der Line–Manager–Task.

Der Main–Task wartet auf keine Signale, sondern reagiert nur auf global gesetzte Flag–Variablen vom HK–Task und TC–Task. Er selber kann an den Histogramm–Task und den Histogramm–Lesetask Signale schicken. Der Histogramm–Task kann mit einem Signal das Auslesen im Histogramm–Lesetask starten; es gibt aber keine Signale in die andere Richtung (der Auslesestatus der Histogramm–Variable wird nicht über Signale, sondern globale Variablen bekannt gemacht). Daher sollte die Gefahr der gefürchteten Verklemmungen in der IASW gebannt sein. Ein etwas abstrakter definiertes Verfahren zur Vermeidung von dead–locks findet sich in [28].

5.8 Fehlerbehandlung

In den bisher beschriebenen Modulen wurde dargelegt, wie die IASW mit **definierten Fehlern** (also Situationen, die eine bestimmte Operation aufgrund äußerer Einflüsse scheitern lassen) umgeht. Die übliche Reaktion ist ein Abbrechen der laufenden Aktion mit dem Verschicken einer Ereignismeldung. Offen blieb aber noch, was in der IASW gegen **intrinsische Fehler**, also echte Softwarefehlern unternommen wird.

5.8.1 Exceptions

Wie in Kapitel 4.2.6 schon angedeutet, gibt es allgemeine Fehlersituationen, die durch sogenannte **Exceptions** (Ausnahmebedingungen) auftreten. Exceptions sind eine Kontrollstruktur der Programmiersprache ADA, die beim Auftreten einer nicht–konformen Operation aus dem gerade laufenden Programmteil herausspringen. Dieser Sprung beendet das Programm, es sei denn, es ist in einen sogenannten Exception–Block geschachtelt, der den Fehler abfängt. Die Struktur eines solchen Exception–Blocks sieht so aus:

```
...
BEGIN
      ...
      irgendwelche Operationen
      ...
EXCEPTION
      Massnahmen zur Fehlerbehebung
END;
```

Treten in den Operationen Fehler auf, so werden direkt die »Maßnahmen zur Fehlerbehandlung« angesprungen. Ein typisches Beispiel ist die Division durch Null, die eine Exception vom Typ »Floatingpoint overflow« erzeugt; die Maßnahme wäre dann eine Fehlermeldung und das Setzen von Standardwerten als Ergebnis. Es ist auch möglich, für jeden Exception–Typ eine eigene Fehlerbehandlung zu schreiben, da ADA dies mit bestimmten Anweisungen unterstützt. Dadurch kann man verschiedene Fehlerursachen individuell behandeln.

Abfangen von Exceptions in der IASW Nicht abgefangene Exceptions beenden das Programm. Dies wäre für die IASW fatal. Zudem darf auch die äußerste Task–Schleife nicht mit Exceptions durchbrochen werden, da sonst der Task nicht mehr laufen kann. Daher müssen *alle* Tasks in Exception–Blöcke eingeschlossen sein. Die Standardreaktion auf das Auftreten einer Exception ist das Abschicken einer Ereignismeldung, wie es in Kapitel 4.2.6 schon beschrieben wurde.[34] Damit eine Fehlerursache eingegrenzt werden kann, ist jedes wichtige

[34] Die Ereignismeldung für Exceptions hat noch ein Feld frei. Es wäre für eine künftige IASW günstig, wenn in diesem Feld die Adresse der Exception (die in dem Programm als Zeichenkette abgelegt ist) eingefügt wird. Damit ist es möglich, den Typ der Exception zu bestimmen. Dies erfordert allerdings etwas Kenntnis über den von TLD implementierten Exception–Mechanismus.

Modul in einen Exception–Block eingeschlossen. Dies betrifft besonders die Verarbeitung von Meßdaten, da hier zum Teil sehr komplexe Operationen durchgeführt werden.[35] Einige Exceptions sind von der IASW selber definiert. In Tabelle 5.12 sind sie aufgelistet. Diese Exceptions sind vor allem als Sicherung

Exception	Beschreibung	Modul
Ex_No_Slew_Time	Slew–Zeit trotz ausgeschaltenem Point–Slew–Automatismus abgefragt.	bcpk_handler
No_Buffer_Available	Versuch, Puffer zu lesen obwohl keine Daten vorhanden sind.	hbr_a_if
No_Block_Available	Versuch, Histogramm–Block von HEPI zu lesen, obwohl keiner im FIFO ist.	histogram_op
Ex_Hsl_Nested_Blocking	Mehrfaches Blockieren der HSL durch einen Task	line_manager
Ex_Lsl_Nested_Blocking	Mehrfaches Blockieren der LSL durch einen Task	line_manager
Ex_Lsl_Was_Not_Blocked	Übertragungsende und LSL war nicht blockiert	line_manager
Ex_Hsl_Was_Not_Blocked	Übertragungsende und HSL war nicht blockiert	line_manager
Ex_Buffer_Not_Empty	Versuch, Histogramme auszulesen, obwohl noch eines gelesen wird.	read_-histogram
Ex_Address_Not_Continuous	Histogramm–Block–Adresse nicht kontinuierlich (S5)	S5
Ex_Address_Not_Continuous	Histogramm–Block–Adresse nicht kontinuierlich (S6)	S6
Ex_Address_Not_Continuous	Histogramm–Block–Adresse nicht kontinuierlich (S7)	S7
Wrong_Table_Parameter	Falsche Tabellenparameter beim Schreiben/Lesen	table_handler

Tabelle 5.12: Exceptions der IASW

gegen eine mißbräuchliche Verwendung der Schnittstellen vorgesehen.

[35] Zu viele Exception–Blöcke können allerdings die Rechenleistung herabsetzen, da jeder Exception–Block etwa 10 μsec Rechenzeit braucht. Sie sollten daher nicht in kleinen, häufig aufgerufenen Routinen verwendet werden.

5.8.2 Task-Overruns

Kritisch ist auch, wenn ein Task nicht kooperativ ist und zu viel Rechenzeit beansprucht. Unter Umständen kann damit das ganze System beeinträchtigt werden, wie auch die Betrachtungen in Kapitel 5.7 zeigten.

Zur Kontrolle wurde die CSSW mit einem Mechanismus ausgestattet, der für zyklische Tasks erkennen kann, ob sie länger als ihre definierte Periode laufen. Dazu wird für jeden Zeittakt (RTC) von einem CSSW–Task mit hoher Priorität geprüft, ob ein zyklischer Task am Ende seiner Periode (die der CSSW bekannt ist) auf das Zeit–Signal wartet oder noch läuft. Im letzeren Falle mißachtet der Task seine Periode, und eine Ereignismeldung wird abgeschickt. Deswegen mussten für die Histogrammtasks einige Vorkehrungen getroffen werden, damit sie sicher keine Periodenüberschreitung durchführen.

Für den HK–Task, der auch zyklisch ist, existiert keine solche Kontrolle, weil er seinen Zyklus mit dem Hardwaresignal `BCP 1` synchronisiert. Ein Überschreiten des acht–Sekunden–Zyklus kann allerdings leicht entdeckt werden, weil dann ein Housekeeping–Paket keine IASW–HK enthält.

5.8.3 Schnittstellenfehler

Der Vollständigkeit halber soll hier noch einmal erwähnt werden, dass bei Nichtbenutzung des Blockierungsmechanismus für die seriellen Schnittstellen im Line–Manager–Task ein Sicherungsmechanismus vorgesehen ist. Vergleiche Kapitel 4.2.1 (letzter Abschnitt) und Kapitel 5.6.2.

5.8.4 Stacküberlauf

Unter keinen Umständen darf der Stapelspeicher, der lokale Prozedurvariablen enthält, überlaufen, weil dann fremde Speicherbereiche überschrieben werden können. Daher ist von der CSSW für den Stapelspeicher jedes Tasks eine dynamische Stackkontrolle vorgesehen. Sie belegt das Ende des Stapelspeichers mit drei Worten eines bestimmten Wertes (5555_{hex}) und prüft regelmäßig nach, ob sich dieser Wert geändert hat. Ist dem so, so wird der entsprechende Task angehalten und eine Ereignismeldung abgeschickt.[36] Dieses Vorgehen ist sinnvoll, aber nicht immer erfolgreich, weil unter Umständen der Stapelspeicherüberlauf auch diesen Mechanismus beschädigen kann. Allerdings würde dies zu einer so starken Beeinträchtigung der Software führen, dass sie wahrscheinlich generell versagen würde. Die Ursache ließe sich dann über ein Auslesen des Stapelspeichers erkennen.[37]

5.8.5 Generelles Software-Versagen (Watchdog)

Was aber, wenn die Software einen totalen Ausfall hat? In der Satellitenbranche hat sich für solche Fälle ein Verfahren etabliert, das in einem regelmäßigen

[36] Verschiedene solche Strategien sind in [5] beschrieben.

[37] Bei einigen Tests der IASW, die zu einem »Absturz« führten, konnte mit dieser Methode der Stapelspeicher untersucht werden und festgestellt werden, dass dies *nicht* die Fehlerursache war.

Rücksetzen eines Hardware–Registers besteht. Wird dieses Register nicht innerhalb einer bestimmten Zeit zurückgesetzt, so erfolgt ein Neustart der Software (Reset). Dieses Vorgehen wird als **Watchdog** bezeichnet, weil quasi ein »Wachhund« auf Hardwareebene die Software kontrolliert. Es gibt verschiedene Arten bezüglich der Form, wer das Watchdog–Register zurücksetzt. Entweder achtet die Hardware auf den Speicherzugriff der Software. Eine andere Möglichkeit ist die, dass die Software selber periodisch in das Watchdog–Register schreibt. In der DPE ist letzterer Weg vorgesehen. Ein zyklischer Task setzt dabei das Watchdog–Register zurück. Wenn dieser Task nicht mehr läuft, führt das zu einem vollständigen Neustart der Software.

Ein »bissiger Wachhund« Ursprünglich war dieser Task der Background–Task der CSSW. Das bedeutete, dass bei jedem Task–Overrun ein Abbruch der Software und Neustart erfolgen konnte, was an sich schon wenig wünschenswert ist.

Für die IASW von IBIS bedeutete dies jedoch, dass der Main–Task im Science–Mode nicht die volle Rechenzeit für die Datenverarbeitung verwenden durfte, sondern genügend Zeit dem Background–Task abgeben musste. Um dies zu bewerkstelligen, musste der nicht–zyklische Main–Task regelmäßig anhalten und warten. Da er selber wenig Kontrolle über seinen Rechenzeitverbrauch hat (er bearbeitet einfach die verschiedenen Ereignisse, wie sie von der HEPI kommen, und wartet gegebenenfalls, wenn mal keine da sind), wurde ein zusätzlicher Task eingeführt: der sogenannte **Main–Control–Task**.

Dieser zyklische Task mit hoher Priorität startete bei bei Beginn eines neuen Zeittaktes (RTC) den Main–Task, und wartete mit Hilfe eines Zeitgebers der DPE 3/4 der verfügbaren Zeit (das sind bei 125 Millisekunden Zeittakt 93.75 Millisekunden), woraufhin der Main–Task wieder angehalten wurde.

Erschwerend kam hinzu, dass der Main–Task **nicht** angehalten werden durfte, wenn er gerade auf ein Signal wartete, weil dieses Signal dann verloren ging. Es musste also für solche Fälle eine Ausnahme gemacht werden, was die Zuverlässigkeit des gesamten Systems nicht erhöhte.[38]

Entschärfung des Problems Bei einem Treffen, an dem einige Entwickler der Instrument–Software und die der CSSW teilnahmen[39], wurde beschlossen, dass der Watchdog nicht mehr vom Background–Task zurückgesetzt werden sollte, sondern von einem Task höherer Priorität. Diese Funktionalität wurde daraufhin in den Minor–Cycle–Task der CSSW[40] verlagert, der fast die höchste Priorität hat. Außerdem wurde in den gleichen Task die schon beschriebene

[38] Diese Methode funktionierte zufriedenstellend. Wenn aber der Main–Task bei ausgeschalteter Abstellfunktion infolge eines Softwarefehlers doch nicht auf ein Signal wartete, konnte der Watchdog zuschlagen. Die Analyse eines Tests wurde durch die Tatsache verkompliziert, dass normalerweise nach einem Watchdog–Neustart sämtliche Informationen über die Ursache des Neustarts gelöscht werden. Deshalb wurde der Watchdog–Reset (nicht aber der Watchdog selber) abgeschalten, und nach einer Aktivierung des Watchdogs konnte der Speicher via Dump–Funktion ausgelesen und analysiert werden.

[39] INTEGRAL–Instrument–Meeting bei GMV, Madrid im November 1998.

[40] CSSW–Version 1.5.

Kontrolle des Task–Overruns eingebaut. Die Benutzung des Main–Control–Task ist damit überflüssig geworden, und die Verarbeitungsgeschwindigkeit der IASW stieg um etwa 25%. Der Watchdog wird nun nur noch bei einem Totalversagen der Software aktiv. Bei kleineren Fehlfunktionen bleibt der Bodenstation genügend Spielraum, um anhand der auftretenden Ereignismeldungen zu entscheiden, wie vorzugehen ist.

Kapitel 6

Beschränkungen im realen System

Bis jetzt wurde beschrieben, wie die IASW mit den an sie gestellten *funktionalen Anforderungen* gerecht wird. Zum Teil wurde auch auf die Vergabe verschiedener Resourcen eingegangen. Es gibt aber noch weitere Anforderungen, die den physikalischen Bedürfnissen der Software Rechnung tragen. Dazu gehört die beschränkte Speichergröße der DPE und die nicht allzugroße Arbeitsgeschwindigkeit ihres Prozessors. Hier soll auf diese Art von Anforderungen eingegangen werden.

6.1 Programmgröße

Da das Programm selber auf E^2–PROMs gebrannt in der DPE fest eingebaut wird, kann es nur eine sehr eingeschränkte Größe haben. Die IASW, mit CSSW und Astres gelinkt, kann maximal 80 kWord belegen[1], die IASW darf nach [3] davon maximal 48 KWord verwenden. Derzeit werden von der IASW 35.5 kWord, also etwa 75 %, belegt.

Damit das Programm nicht zu groß wird, mußte auf einen nicht zu extensiven Programmierstil geachtet werden. Insbesondere vergrößerte eine Initialisierung mit einer Zuweisung, wie sie in ADA bequem zu machen ist, umfangreiche Felder enorm, weil dazu die Initialisierungswerte als Konstante definiert werden. Daher musste die Zuweisung durch eine **dynamische Initialisierung** mit sequentieller Einzelwertzuweisung ersetzt werden. Als Beispiel sollte folgende Initialisierung nicht verwendet werden:

```
PACKAGE BODY Beispiel IS

TYPE T_TABELLE IS ARRAY(1..10000) OF INTEGER;

TABELLE : T_TABELLE := (OTHERS => 0),--initialisiert Tabelle mit 0
                                     --!!! ZU VIEL SPEICHER !!!
```

Besser ist:

[1] 20 PROMs mit je 4 KWord (8 kByte).

```
PACKAGE BODY Beispiel IS
...
TABELLE : T_TABELLE;
...
BEGIN
   FOR I IN (1..10000) LOOP
     TABELLE(I) := 0;
   END LOOP;
END Beispiel;
```

6.2 Speicheraufteilung

Die physikalische Speichergröße der DPE mit zwei MegaByte ist mehr als ausreichend. Die Schwierigkeiten liegen in der Art, wie der 1750A–Prozessor auf diesen Speicher zugreift. Denn der Prozessor kann nur auf 64 kWord Daten gleichzeitig zugreifen, und muss den Adresszustand wechseln, wenn er anderen Speicher verwenden will. (Siehe auch Kapitel 2.4.) Dieser sogenannte **Transit** kostet aber Zeit.[2] Aus diesem Grund sollten zusammengehörige Module möglichst im gleichen Adresszustand liegen, und der Transit sollte nur für bestimmte Funktionen verwendet werden.

Speicheraufteilung in Knoten Ein weiteres Problem eines Transits ist, dass Variablen, die in einem Adresszustand sichtbar waren, es im anderen nicht notwendigerweise mehr sind. Eine Variable übersteht nur dann einem Transit, wenn sie in einer Speicherseite (Page) liegt, die zu beiden Adresszuständen gehört.

Dies wird von der Implementierung des ADA–Compilers/Linkers unterstützt, der ein hierarchisches Knotenmodell mit einer Baumstruktur für Speicherbereiche benutzt. Häufig verwendete Module kommen in den obersten Knoten und sind damit für alle Adresszustände sichtbar, während weiter unten liegende Knoten nur für die Adresszustände erreichbar sind, die am selben Knoten hängen. Die untersten Knoten entsprechen den verschiedenen Adresszuständen. Die Zuordnung der Module in den verschiedenen Knoten zu den Speicherseiten übernimmt der Linker. Bild 6.1 möge dies verdeutlichen. Bei einem Transit gilt dann die Regel, dass nur auf Variablen eines gemeinsamen Knoten zugegriffen werden kann. Solche Variablen sind in Speicherseiten untergebracht, die auch in verschiedenen Adresszuständen auf den gleichen physikalischen Speicher verweisen.

Man muss allerdings darauf achten, dass nicht zu viele Module in die oberen Knoten gelegt werden, weil dies den Speicherbereich, der gemeinsam verwendet wird, erhöht und damit den durch einen Transit gewonnenen physikalisch verfügbaren Speicher verkleinert. Im Extremfall haben alle Adresszustände den gleichen physikalischen Speicherbereich belegt, und ein Transit hätte keinen Effekt.

[2] Etwa 68 μsec.

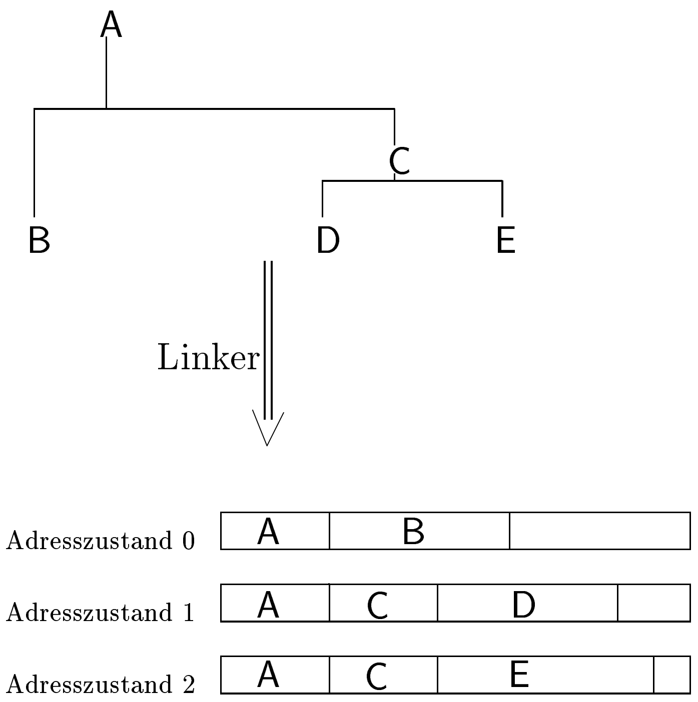

Abbildung 6.1: Knotenstruktur und ihre Repräsentation

Allgemeine Vorgaben bei der Speicheraufteilung In der Software (IASW und CSSW) gibt es einige Variablen und Prozeduren, die in allen Adresszuständen sichtbar sein sollten. Dazu gehört etwa Exception–Variablen oder die Assemblerroutine zur Prüfsummenberechnung. Um Variablen beliebiger Größe in mehreren Adresszuständen sichtbar zu machen, wurden für die IASW folgende Vorgaben gemacht:

1. Der **Stapelspeicher** jedes Tasks ist **global sichtbar**, d.h. kann in allen von ihm erreichbaren Adresszuständen benutzt werden.

2. Variablen, die einen Transit überdauern sollen, werden auf dem Stapelspeicher abgelegt. Sie dürfen nicht statisch in einem Modul deklariert werden.

Das bedeutet, dass Variablen, die bei einem Transit erhalten bleiben sollen, *lokal* in einer Prozedur deklariert werden. Der *Vorteil* ist, dass die Module nicht ihre Struktur ändern müssen, indem sie etwa eine globale Variable in einem überall sichtbaren Modul ablegen. Sie müssen nur obige Regeln befolgen. Außerdem spart diese Methode an global sichtbarem Speicherplatz, weil der Stapelspeicher mehrfach verwendet werden kann, da jeder parallele Aufruf einer Prozedur den gleichen Speicher benutzt.

Nachteilig hingegen ist, dass es keinerlei automatische Kontrolle gibt, ob eine Variable sichtbar oder nicht ist. Bei statisch abgelegten Variablen, die von einer bestimmten Stelle aus nicht sichtbar sind, gibt der Linker eine Warnung aus. Zudem muss sehr sorgfältig auf die Größe des Stapelspeichers geachtet werden.

Speicheraufteilung der IASW Die beste Speicheraufteilung wäre, wenn die gesamte IASW in einem Adresszustand läuft, und nur gelegentlich bei Bedarf auf andere Speicherteile zugreift. Dies ist weitgehend geschehen; der Hauptteil der IASW und die gesamte CSSW befindet sich im Adresszustand 0. Ausgelagert sind zunächst nur die größeren Tabellen, wobei die Tabelle zur Energiekorrektur von CdTe–Photonenereignissen mit 340 kWord auf fünf Adresszustände verteilt werden musste. Weiterhin ist auch ein Adresszustand für die Kontexttabellen verwendet worden. Leider passt der Rest der IASW nicht in einen Adresszustand von 64 kWord. Deshalb befindet sich die gesamte Histogramm–Steuerung in Adresszustand 1. Das ist nicht besonders nachteilig, weil die Steuerung und Bearbeitung von Histogrammen zyklisch mit hoher Priorität am Stück läuft. Zudem beschränken sich die Verbindungen zum Rest der IASW auf die Start/Stop–Signale und einige Housekeeping–Werte. In Tabelle 6.1 sind die Programmteile und ihre Adresszustände aufgeführt.

Adresszustand	Programmteil
0	CSSW, HK–Task, TC–Task, Einzelereigniss–Bearbeitung, alle Schnittstellen nach außen
1	Histogramm–Tasks, Histogramm–Bearbeitung
2	Kontext–Tabellen und Verwaltung
3–7	CdTe–Energie–Korrektur–Tabelle

Tabelle 6.1: Adresszustände in der IASW

6.3 Zuteilung der Rechenleistung

Die von der DPE verfügbare Rechenleistung ist gering; sie beträgt etwa 1 MIPS.[3] Bei einer Rate von 2000 Einzelereignissen pro Sekunde sind das etwa 500 Maschineninstruktionen pro Ereignis, was nicht viel ist.[4] Damit die **Zuteilung der Rechenleistung** für verschiedene Datenverarbeitungsaufgaben möglichst flexibel ist, kann man entweder die Bearbeitung in mehrere Tasks mit verschiedenen Prioritäten aufteilen[5], oder die Rechenzeit wird mit einem dynamischen Konzept bei der Bearbeitung verwaltet.

In der IASW ist letzterer Weg mit den Schwellwerten der Telemetrie gewählt worden. Damit ist eine Verteilung an Rechenzeit möglich, weil für jeden Tele-

[3] Die Taktfrequenz ist 13 MHz. Als ein Prozessor mit typischer CISC–Architektur braucht dieser etwa 10 Takte pro Maschineninstruktion.

[4] Im folgenden geht es nur um die Bearbeitung von Einzeereingissen im Science/Diagnostic–Mode; die übrigen Steuerungsaufgaben der DPE und die Histogrammbearbeitung sind nicht zeitkritisch und daher auch nicht Teil der folgenden Betrachtungen.

[5] Dies macht bei genauerer Analyse wenig Sinn. Denn dann würde der Task mit der höchsten Priorität zunächst *alle* ihm zugeteilten Ereignisse bearbeiten, um dann den eventuell sehr kleinen Rest an Zeit den anderen Tasks zu überlassen. Möglich, aber wohl zu langsam wäre vielleicht ein Konzept mit zyklischen Tasks, die mit unterschiedlichen Perioden laufen.

metrietyp je nach Pufferfüllstand die Bearbeitung an– oder ausschaltet wird. Telemetriepakete mit hohen Schwellwerten sind bevorzugt, weil sie bei hohen Datenraten mehr Telemetrieresourcen zugeteilt bekommen. Telemetriepakete mit niedrigen Schwellwerten sind nicht absolut vernachlässigt, weil nach ihrem Abschalten die gesamte Telemetrieerzeugung und damit auch der Pufferstand wieder sinkt.

Nachteilhaft ist allerdings, dass die Telemetrie unter Umständen zerhackt wird, was eine Auswertung der wissenschaftlichen Daten erschweren kann, besonders, wenn es auf eine Zeitreihenanalyse ankommt. Problematisch ist auch, ein absolutes Verhältnis der Telemetrieerzeugungsraten zu definieren, weil diese sehr von der Erzeugungsrate der verschiedenen Einzelereignisse abhängt. Beispielsweise bedeutet ein Schwellwertverhältnis von 2:1 nicht eine Telemetrierate von 2:1, wenn die Erzeugungsrate verschieden ist.

Kapitel 7

Die Software-Entwicklung

Ein Programm, das zuverlässig eine vorgegebene Aufgabe erfüllen muss, hat viele Entwicklungsschritte zu durchlaufen, bevor es eingesetzt werden kann. Die Vorgehensweise bei Entwurf, Umsetzung und Prüfung der IASW wird im folgenden dargestellt.

7.1 Design

Zuerst wurde versucht, aus den Anforderungen einen grobstrukturierten Entwurf der Software zu erstellen. Dies war schwierig, weil sich die Anforderungen im Laufe der Zeit verändert haben.[1] Dies liegt zum Teil an der Komplexität des IBIS–Instrumentes mit seinen Subinstrumenten, die in verschiedenen Ländern gebaut werden. Da sich auch diese Subinstrumente noch in Entwicklung befinden, war und ist es nicht einfach, sich ad hoc auf eine verbindliche, alles umfassende Architektur des Systems zu einigen. Gerade das Protokoll der Low speed Serial Line und das der Tabellen haben viele Änderungen erfahren. Auf der anderen Seite gab es Probleme, die sich aus den verschiedenen physikalischen Beschränkungen ergaben. So war nicht von Anfang an klar, wie am besten mit der eingeschränkten Rechenleistung der Datenverarbeitung umgegangen werden soll.

Software Design Aus diesem Grunde konnte anfangs nur ein grobes Gerüst der Software–Architektur der IASW erstellt werden. Es wurde versucht, die Software in möglichst abgeschlossenen Module aufzuteilen. Besonders wurde auf die Schnittstellen geachtet, da bei der Entwicklung mehrere Personen beteiligt waren, deren Softwareteile zusammenarbeiten sollten. Dazu wurden in einem objektorientierten Ansatz zunächst die Gemeinsamkeiten der Schnittstellen und dann die genauen Einzelheiten festgelegt. Eine Beschreibung findet sich in [36], die der Implementierung der IASW allerdings nur grob entspricht.

[1] Die ursprünglichen Anforderungen sind in [3] festgehalten.

7.2 Die Programmiersprache ADA

Für die Implementierung der IASW war die Programmiersprache ADA 83 vorgegeben. In Sonderfällen wurde auch in der 1750A–Assemblersprache programmiert. Diese Sprache wurde im Auftrag des Departments of Defense entwickelt, als sich eine Software–Krise militärisch genutzter Programme anbahnte. Es stellte sich heraus, dass eine Pflege und Portierung von Programmen wegen eines fehlenden strengen Standards nahezu unmöglich geworden war. Die Programmiersprache ADA behob dieses Manko, indem sie zum einen auf eine ausgefeilte, für alle Implementierungen verbindliche Spezifikation aufbaut[2], zum anderen mit einer strengen Typisierung Schnittstellenprobleme möglichst schon während der Übersetzungsphase eines Programmes erkennt. Hinzu kommt, dass man bei ADA eine Software mit Hilfe von **Packages** leicht in verschiedene Module aufteilen kann. Aus diesen Gründen ist sie auch in der Raumfahrt sehr beliebt.

Die Sprache ähnelt der später entstandenen Sprache *Pascal*, es kommt aber unter anderem das schon beschriebene Konzept der Exceptions hinzu, mit dem auf Laufzeitfehler eine definierte Reaktion erfolgt. Eine Besonderheit von ADA ist die Möglichkeit, Parameter einer Funktion über ihren Namen anzusprechen, sowie ein umfangreiches Konzept zur Variableninitialisierung, die sogenannte »Elaboration«. Die Sprachdefinition findet sich z.B. in [16]. Eine Einführung bietet [23, 18].

Das ADA–Compilersystem Für die Entwicklung der IASW auf IBIS wurde ein ADA–Compilersystem der Firma TLD verwendet, das auf den 1750A–Prozessor der DPE zugeschnitten ist. Das ADA–Compilersystem selber läuft auf einer SUN–Workstation, gehört damit also in die Klasse der Cross–Compiler. Die verschiedenen Module (ADA–Packages) bestehen aus je zwei Dateien, von denen eine die Schnittstellen beschreibt (die »Specification«), während die andere die konkrete Implementierung beinhaltet (der »Body« des Moduls). Jedes einzelne Modul, sogar jede Datei kann getrennt compiliert werden. Die Compilierung ist optimiert. Optional kann der Compiler Informationen über den erzeugten Code ausgeben, was bei einer Fehlersuche zuweilen nützlich ist.

Alle compilierten Module werden in einem **Link–Lauf** zu dem fertigen Programm zusammengesetzt. In einem speziellen **Link–File** ist neben den verwendeten Modulen (mit Pfadangabe) die schon beschriebene Speicheraufteilung in Knoten festlegt. Mit dieser Datei ist es möglich, die Adressbelegung einzelner Variablen zu bestimmen, was für eine Mehrfachbelegung im Falle der Parametertabellen sehr nützlich war.[3]

Während des Linkens kann eine Angabe über die Speicherbelegung in einem sogenannten **Map–File** ausgegeben werden. Mit Hilfe dieser Datei kann eine mögliche Fehlerursache analysiert werden. Die Ereignismeldungen der CSSW geben die Adresse einer Excpetion an, die mit der Map–Datei aufgeschlüsselt

[2] Es lohnt sich, die Spezifikationsentwicklung von ADA zu lesen, da diese nicht in einem Anlauf, sondern in mehreren Schritten von einer recht allgemeinen Festlegung auf alle denkbaren Fälle erweitert worden ist. Siehe dazu [26].

[3] Bei jeder Mehrfachbelegung gibt der Linker eine Warnung aus. Aus diesem Grund erscheinen beim Übersetzen der IASW 57 solcher Warnungen, die aber keine Fehler darstellen.

werden kann. Diese Datei ist unverzichtbar, wenn auf eine bestimmte Variable via Patch/Dump zugegriffen werden soll, weil über sie die Adresse einer Variablen ermittelt werden kann.

Debugger/Simulator Für eine Fehleranalyse ist ein sogenannter **Debugger** vorgesehen. Er führt jeden Maschinenbefehl eines compilierten Programms aus, gewährt Zugriff auf alle in diesem Programm definierten Variablen und hält an gewünschten Programmstellen an. Der Debugger des TLD Compilersystems läuft auf einer SUN Workstation und erlaubt eine Simulation komplexer Prozesse, da ein externes Programm alle Ein– und Ausgaben des Prozessors abfangen kann. Ein solches externes Programm ist von der Firma GMV mit der CSSW ausgeliefert worden, welches alle Funktionen der DPE (Telekommandos, Telemetrie, analoge, digitale und serielle Leitungen) nachbilden kann.

Allerdings ist die Simulation nicht allzu schnell; zwei Sekunden simulierte Zeit entsprechen 10 Minuten realer Zeit. Dennoch konnten viele Teile der IASW mit diesem Simulator ausgestestet werden, da der Debugger Einblicke in die laufenden Vorgänge ermöglicht, die auf der echten DPE nur schwer oder überhaupt nicht zu bekommen waren.

Schwierigkeiten im Umgang mit dem Compiler Unter besonderen Umständen versagt der Compiler und erzeugt definitiv falschen Code [27]. In diesen Fällen muss der Algorithmus umgeschrieben werden.

Es gibt aber auch Situationen, die keinen Compilerfehler darstellen, aber bei Nichtberücksichtigung zu fatalem Fehlverhalten führen können. Dazu gehört, dass der Compiler/Linker nicht erkennen kann, ob eine Variable, die einer Prozedur als Parameter übergeben worden ist, im aktuellen Adresszustand sichtbar ist. Einerseits überlässt es die ADA–Spezifikation der Compiler–Implementierung, welche Parameter eines Prozeduraufrufes als Wert (≫Call by Value≪) und auf welche Parameter über die Adresse zugegriffen wird (≫Call by reference≪). Anderseits ist gerade diese Unterscheidung bei der Implemetation des TLD–Compilers wichtig. Denn ersteres ist unproblematisch, da der übergebene Parameter kopiert wird und die Kopie innerhalb der Prozedur sichtbar ist (entweder befindet sie sich in einem Prozessorregister oder auf dem Stapelspeicher). Schwierig ist es, wenn eine Variablenadresse übergeben worden ist und die Prozedur einen Transit vorgenommen hat. In diesem Fall verweist die Variablenadresse nach wie vor auf einen logischen Speicherplatz, der nun aber in einem anderen Adresszustand nicht mehr die übergebene Parametervariable enthalten muss.

Die Implementierung des TLD–Compilers legt fest, dass auf alle Parameter, die keine Skalare (Integer–Variablen und Real–Variablen) sind, mit Adressbezug (≫per reference≪) zugegriffen wird [31]. Daher müssen solche Variablen, sofern die Prozedur einen Transit durchführt, global sichtbar sein; siehe auch Kapitel 6.2.[4]

[4]Besonders kniffelig ist es, wenn einer Prozedur mit Transit ein Feld unbeschränkter Größe als Parameter übergeben werden soll. Denn der Compiler legt für solche Felder eine Struktur an, die unter anderem die tatsächliche Größe einer Feldvariablen enthält. Zuweilen möchte man aber auch Felder der Länge Null übergeben (nämlich Kommandos an das Detektor–

Eine weitere Schwierigkeit betrifft den **Stapelspeicher** (Stack), auf dem lokale Variablen einer Prozedur abgelegt werden können. Er sollte nicht zu groß sein, weil er in der IASW global sichtbar ist und damit den Speichergewinn bei einem Transit verkleinert. Auf der anderen Seite darf er auch bei maximaler Prozedurverschachtelung nicht überlaufen, d.h. andere Speicherbereiche überschreiben.

Die erforderliche Stapelspeichergröße der verschiedenen Tasks kann mit Hilfe des TLD Compilersystems ermittelt werden. Der Linker kann alle Prozeduraufrufe während des Linkens verfolgen und aus der jeweiligen Parametergröße die Stapelspeichergröße bestimmen. Vorraussetzung ist allerdings, dass

- Keine Prozedur rekursiv (selbstaufrufend) ist.
- Die Parametergröße der Prozeduren fest ist.

In der IASW findet keine Rekursion statt. Allerdings ist die Parametergröße für die Kommandos an die Subinstrumente nicht konstant (die Kommandos können ja verschiedene Länge haben). Die maximale Kommandogröße beträgt laut Spezifikation 66 Word. In diesem Falle muss also die Stapelspeichergröße um diesen Betrag erhöht werden.[5] Siehe auch [30, 29].

7.3 Entwicklungsschritte

Die Entwicklung der IASW lief in mehreren Schritten ab, in denen sie erweitert, verbessert und getestet wurde.

Test-Programme Der erste Schritt bestand im Schreiben kleiner Testprogramme, die zunächst einmal mit dem Multitaskingsystem, den Schnittstellen der DPE und der Softwareerstellung vertraut machen sollten.

Im allererste Testprogramm wurde versucht, ein bestimmtes Housekeeping–Paket abzuschicken. Ein zweites Testprogramm erzeugte auf ein Telekommando hin ein Telemetriepaket. Ein weiteres las jede Sekunde (synchronisiert mit dem BCP 1–Signal) die An–Bord–Zeit (OBT) aus und schickte sie als Telemetriepaket ab. Gerade dieses letzte Programm half, das Signalsystem der CSSW/Astres zu verstehen.

SIASW–Ein Prototyp Für die grundlegenden Funktionen wurde eine Beispiel-IASW (**SIASW** — Sample–IASW) geschrieben, die als Basis der echten IASW dienen sollte. Sie enthielt schon die Umschaltung zwischen verschiedenen Modi und das Weiterschicken externer Telekommandos. Außerdem

Interface ohne Parameter). In diesem Fall ist die beschriebene Struktur konstant und wird vom Compiler statisch, also nur für den aktuellen Adresszustand sichtbar angelegt. *Nach* dem Transit ist sie aber nicht mehr sichtbar, und die Feldlänge stimmt nicht mehr. An die beschriebene Struktur hat man über die Hochsprache keinen Zugriff. Als Abhilfe gab es nur die Möglichkeit, der Prozedur ein Feld variabler Größe zu übergeben, die im konkreten Falle auf Null initialisiert ist. Dieses Problem taucht auf, wenn auf das Modul DETECTOR_INTERFACE mit Transit zugegriffen werden soll.

[5] Es empfiehlt sich, etwas Spielraum zu lassen, weil Astres selbst etwas Stapelspeicher (etwa 10 Word) benötigt.

konnte sie bald Daten via High speed Serial Line A einlesen und als Diagnostik–Telemetriepakete abschicken. Damit war eine Steuerung der HEPI für Testzwecke relativ früh möglich. Einen Science–Mode gab es allerdings noch nicht.

Schrittweise Verfeinerung Im Laufe der Zeit kam die Implementierung aller Telekommdos und die komplette Tabellenverwaltung hinzu. Auch die zusätzlichen Funktionen des Science–Mode, der die HEPI–Konfiguration selbstständig steuert, wurde ausgehend vom vorhandenen Diagnostic–Modus geschrieben. Viele Module wurden dabei mehrfach geändert und teilweise komplett neu entworfen. Die meisten Veränderungen waren jedoch bei der Entwicklung der Histogrammsteuerung notwendig, da die komplexe Verflechtung von Integration und Auslesen nicht mit einem einfachen Ansatz erledigt werden konnten. Generell wurde für jedes Modul zunächst die Schnittstelle definiert, um dann die Funktionalität des Moduls schrittweise mit Tests im Simulator zu verfeinern.

7.4 Arbeiten im Team

Die Software wurde und wird im Team entwickelt. So ist unter anderem die komplexe Verarbeitung der wissenschaftlichen Daten, die in der vorliegenden Arbeit fast vollständig ausgeklammert wurde, von Martin Stuhlinger geschrieben. Gerade diese Module sind bezüglich der Geschwindigkeit hoch optimiert, um eine Verarbeitung aller Einzelereignisse zu gewährleisten. Dieser Softwareteil ist in [27] ausführlich beschrieben.

Patrick Risse beteiligte sich beim Entwurf der IASW und setzte die Grundstruktur der Module um, die dem konkreten Aufbau der IASW dienten.

Die Spezifikation ist im wesentlichen von Reiner Volkmer, dem Projektleiter des INTEGRAL–Teams am IAAT, im Austausch mit den verschiedenen, an IBIS beteiligten Gruppen, erarbeitet worden.

Einige Module (Zusätzliche Housekeeping, Noisy–Pixel–Behandlung, Operationen der An–Bord–Zeit (OBT)) sind in der Firma LABEN entwickelt worden.

7.4.1 Software–Verwaltung

Die Software eines solchen Teams muss so verwaltet werden, dass Änderungen, die von einem Entwickler vorgenommen wurden, protokolliert und den anderen Beteiligten zugänglich gemacht werden können. Dazu wurde das Versionskontrollsystem **SCCS** (Source Code Control System), das auf den meisten UNIX–Plattformen verfügbar ist, verwendet. Es übernimmt die Versionskontrolle einzelner Dateien und erlaubt den Entwicklern, beliebige Änderungen vorzunehmen und auszutesten, bevor sie zum Gesamtsystem hinzugefügt werden. Siehe auch [10].

Make Da ein kompletter Compilerlauf recht lange (bei der derzeitigen IASW bis zu 20 Minuten) dauern kann, ist es wünschenswert, nur die geänderten Dateien zu compilieren. Das UNIX–Programm **make** unterstützt dies, indem es aus Angaben der Abhängigkeit aller Dateien die notwendigen Module übersetzt.

Diese Abhängigkeiten, die angeben, welches Modul auf welches zugreift, sind in einer Datei namens ≫makefile≪ festgehalten.

Das Bestimmen der Abhängigkeiten von inzwischen über hundert Dateien ist jedoch aufwendig, so dass ein UNIX–Script namens **genmake** entwickelt wurde, welches die Abhängigkeiten aus den ADA–Dateien ermitteln und ein komplettes ≫makefile≪ erstellen kann.[6] Vorraussetzung ist, dass alle Dateien der IASW im aktuellen Verzeichnis liegen, kleingeschrieben und mit der Endung *.ada versehen sind, und dass die Specifikation vom Body eines ADA–Packages mit dem Suffix ≫_≪ unterschieden ist.

7.5 Verifikation

Nach der Implementierung musste die IASW gegen ihre Spezifikation validiert werden, d.h. es muss geprüft werden, ob sie so arbeitet, wie von ihr verlangt worden ist. Häufig geschieht dies durch Testen. Auf Modulebene wurden daher im Simulator viele verschiedene Testfälle der IASW durchgespielt.

Testmuster Für die Eingabe von Testmustern, die besonders die Einzelereignisse betreffen, sind am IAAT verschiedene Programme entwickelt worden. Dazu wurde von Robert Weiss einen Detektorsimulator geschrieben, der verschiedene Einzelereignisse erzeugen kann [37]. Weiterhin hat Oliver Decker einen HEPI–Simulator entwickelt, der alle Verarbeitungsschritte der HEPI nachbilden kann und aus den Detektorereignissen Daten im HEPI–Format erzeugt [4]. Für die Interpretation der Ausgabe des DPE–Simulators ist von Patrick Risse ein Programm entwickelt worden, welches die Telemetriepakete sortieren und die Einzelereignisse herausfiltern kann.

Zeitmessungen Die Datenverarbeitung im Satellit ist sehr zeitintensiv. Deshalb musste eine Testmethode entwickelt werden, mit der die Verarbeitungsgeschwindigkeit der verschiedenen Programmteile gemessen werden kann. An der echten DPE ist eine Zeitmessung möglich, sofern ein geeignetes Gerät die Vorgänge innerhalb der DPE analysieren kann. Leider ist z.Z. ein solches Gerät nicht verfügbar, so dass eine Zeitmessung derzeit nur im Simulator möglich ist. Der Simulator kann aus der Zahl ausgeführter Maschinenbefehle und der Taktfrequenz des Prozessors die simulierte Zeit berechnen. Für eine einfachere Handhabung wurde ein kleines UNIX–Script (SHOW_TIME) geschrieben, das aus den Zeitangaben in der Simulator–Logdatei die Laufzeit berechnen kann. Für eine Untersuchung der Multitaskingvorgänge wurde ein Haltepunkt an die Stelle der Task–Umschaltung gesetzt, und aus der Zeitangabe die Task–Laufzeiten bestimmt und in einem Diagramm ausgegeben. Die Konvertierung kann mit einem UNIX–Script namens SHOW_TASKS vorgenommen werden.

Testlauf auf der DPE Um ein Verhalten der Software in der Realität zu erhalten, musste sie auch auf der realen DPE extensiv getestet werden. Dazu stand

[6] Dieses Script durchläuft alle ADA–Dateien und liest die ≫WITH Package≪ Anweisungen.

Abbildung 7.1: Satelliten–Simulator als Testumgebung

eine reale DPE zur Verfügung, die von einem komplexen Satelliten–Simulator (SIS) und einer HEPI die Funktionen der IASW testbar machten. Als Eingabe an Einzelereignissen gab es ein Detektorsimulator, später dann auch Elektronikeinheiten mit der vollen Detektorfunktionalität. Letztere Konfiguration diente für die Abnahmetests der IASW.

Ein Monitor-Programm in der DPE Um einen Einblick in die Vorgänge der echten DPE zu bekommen, gab es ein Programm, das zur IASW/CSSW gelinkt werden konnte. Mit diesem sogenannten **Monitor** konnte über eine spezielle serielle Leitung kommuniziert werden. Damit war es unter anderem möglich, das komplette Programm in den Speicher der DPE zu laden, was sonst immer mit relativ großem Aufwand verbunden ist (die IASW muss auf E^2PROMS gebrannt und in einem Schuber der DPE ausgewechselt werden). In beschränktem Ausmaß sind mit dem Monitor auch Debugger–Befehle verfügbar.

Korrektheitbeweise Für verschiedene Fälle (Zeitverhalten etc.) reichen einfache Tests nicht immer aus. Dann müssen andere Prüfverfahren verwendet werden (Korrektheitsbeweise, Gegenprüfung). Dies betrifft besonders die Implementierung der Schnittstellenverwaltung, in der ein Fehler auftreten kann, aber nicht muss, je nachdem, ob sich gerade zwei Tasks gleichzeitig um Zugriff auf eine serielle Leitung bewerben. In diesem Falle wurde, wenn auch nicht mathematisch streng, versucht, einen Beweis der Korrektheit des Algorithmus zu führen. Auch die globale Sichtbarkeit von Variablen während eines Prozedur–

Transits stellt eine nicht testbare Situation dar, die aber in einem »Walk–through« der IASW gemeistert werden kann. Dabei werden alle Prozeduren, die einen Transit vornehmen, aufgeschrieben und deren Parameter auf globale Sichtbarkeit untersucht.

Integrationstests Nachdem die einzelnen Module sich als funktionsfähig erwiesen hatten, mussten sie gelinkt werden, wobei schon einfache Schnittstellenfehler zutage traten. Weiterhin wurde versucht, die Funktionalität der Abläufe zwischen den Modulen im Simulator auszutesten. Schliesslich ist die komplette IASW auf der DPE geprüft worden.

Abnahmetests Für die verschiedenen Modelle des Satelliten gab es Abnahmetests, die teilweise von Mitarbeitern der Firma LABEN entworfen und durchgeführt wurden. Die IASW hatte viele Mängel und Unstimmigkeiten, die aber weitgehend verstanden sind und in einer künftigen Version behoben werden.

Zusammenfassung und Ausblick

Softwareentwicklung dauert stets länger als geplant; diese Regel gilt auch, wenn sie in die Planung einbezogen ist.

Im Fall der IASW–Entwicklung hat sich dieser Grundsatz bestätigt. Mit dem Design wurde im Dezember 1997, mit der Implementierung im Februar 1998 begonnen, und die Software sollte spätestens im Juni 1998 fertiggestellt sein, da dies der Abgabetermin war.

Die ersten Abnahmetests der IASW mit eingeschränkter Funktionalität fanden im September 1998, also gut drei Monate später statt. Die Mängelberichte waren umfangreich. Im März 1999 konnte die IASW eine volle Funktionalität bieten (allerdings noch mit mehreren Softwarefehlern).

Wieso hat die Implementierung der IASW den Zeitrahmen um fast das vierfache gesprengt? Im Rückblick lässt sich feststellen, dass ich viel Zeit für das Schreiben von verschachtelten und umfangreichen Softwareteilen aufbringen musste. Beispielsweise ist die Telekommandobehandlung sehr umfangreich und enthält — zumindest, was die Steuerung betrifft — das größte Modul der IASW. Diese Module haben entsprechend während der Implementierung viele Änderungen erfahren. Auf der anderen Seite ist die Telekommandobehandlung umfangreich, aber nicht besonders kompliziert oder verschachtelt, weil die verschiedenen Funktionen in viele unabhängige Unterroutinen aufgeteilt werden können.

Im Gegensatz dazu ist die Verwaltung der seriellen Schnittstellen mit Semaphores sehr verschachtelt. Dennoch konnte die Entwicklung im wesentlichen im Mai 1998 abgeschlossen werden, weil sich der Algorithmus auf ein einziges Modul beschränkt. Auch das Modul zum Lesen von Einzelereignissen ist relativ überschaubar, so dass schon im März 1998 eine Verarbeitung von Ereignissen möglich war.

Schwieriger war die Implementierung der Kontexttabellen, weil es zwar eine gute Beschreibung des Protokolls gab, HEPI und die drei anderen Detektoren jedoch spezielle Anforderungen stellten. Nicht zuletzt wegen vieler Missverständnisse, die zum Teil erst im Test mit allen Subinstrumenten behoben werden konnten, hat sich die Entwicklung bis Februar 1999 stark verzögert.

Das sicherlich umfangreichste und zugleich verschachtelste Softwareteil betrifft die Histogrammbearbeitung. Schon die genaue Angabe, wann Histogramme integriert und ausgelesen werden sollten, machte großes Kopfzerbrechen, und das Konzept ist oft umgearbeitet worden. Auch die Implementierung, die die Zeitabläufe koordinieren muss, hat viele Revisionen erfahren. Selbst das Testen im Debugger erforderte einen umfangreichen Testplan, da der komplette Ablauf über große Zeiträume von 1000 Sekunden (die Simulation dauert rund 3.5 Tage reale Zeit) verteilt ist. Insofern kann es als großer Erfolg gewertet werden, dass im März 1999 bei den Abnahmetests Histogramme vollständig integriert und ausgelesen werden konnten.

Mit diesem „Second Light“[7] ist gezeigt, dass für das IBIS -Experiment die Kette von den Mess-Ereignissen im Detektor bis zur Datenauswertung auch für komplexe Bilder (denn was sind Histogramme anderes als Bilder) geschlossen ist. Man darf also auf Bilder des INTEGRAL–Satelliten mit hoher Auflösung im Gammabereich gespannt sein. Für diesen Erfolg möchte ich mich bei meinem Team — Martin Stuhlinger, Reiner Volkmer (Projektleiter), Patrick Risse, Claus Dreischer und Nikolai von Krusenstiern (HEPI) — , bei Herrn Staubert, der dieses Projekt mit viel Geduld begleitet hat und nicht zuletzt bei Mike Rennie von GMV, dem Entwickler der CSSW, der als Mentor im Hintergrund viele gute Ratschläge für die Entwicklung der IASW beigesteuert hat, bedanken.

[7]Das „First Light“ war die erfolgreiche Verarbeitung von Einzelereignissen im Oktober 1998.

Abkürzungen

ACC	Attitude Control Computer
ADA	Programmiersprache ADA
APID	Application Identifier
bps	Bit per Second
BCP	Broadcast Pulse
BCPK	Broadcast Packet
Byte	Speicherzelle von 8 Bit
CCD	Charge Coupled Device
CDMU	Central Data Management Unit
CdTe	Cadmium Telluride Detektor (ISGRI)
CID	Command Identifier
CISC	Complex Instruction Set Computer
CRC	Cyclical Redundancy Code
CRF	Context Restore Flags
CSF	Context Save Flags
CSSW	Common Service Software
CsI	Caesium Iodide Detektor (PICsIT)
cts	Counts per Second
DHS	Data Handling System
DMA	Direct Memory Access
DPE	Data Processing Electronics
ECR	Electrical Configuration Register
FCFOV	Full coded field of view
FID	Function Identifier
FIFO	First In, First Out (Puffertyp)
HBR	High Bit Rate Line, HSL
HEPI	Hardware Event Preprocessor of IBIS
HK	Housekeeping Packet
HSL	High Speed Serial Line
IAAT	Institut für Astronomie und Astrophysik Tübingen
IASW	Instrument Application SoftWare
IBIS	Imager on Board INTEGRAL Satellite
INTEGRAL	INTErnational Gamma–Ray Astrophysics Laboratory
ISGRI	INTEGRAL Soft Gamma Ray Imager
JEM–X	Joint European Monitor of X–Rays
LBR	Low Bit Rate Line, LSL
LSB	Least Significant Byte

LSL	Low Speed Serial Line
LSW	Least Significant Word
LUT	Look up Table
MCE	Module Controlling Electronics
MIPS	Mega–Instruction Per Second
MSB	Most Significant Byte
MSW	Most Significant Word
OBT	On Board Time
OBDH	On–Board Data Handling
OEM	On–Event Message
OMC	Optical Monitoring Camera
PICsIT	Pixellated Imaging CsI Telescope
PROM	Programmable Read–Only Memory
PPM	Photon by Photon Mode
PST	Polling Sequence Table
RTC	Realtime Clock
SIS	Spacecraft Interface Simulator
SPI	Spectrometer on INTEGRAL
TC	Telecommand
TID	Task Identifier
TM	Telemetry Packet
Word	Speicherzelle von 16 Bit
VETO	Veto shield

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] S. Bergeson-Willis et al. INTEGRAL- Report on the Phase A Study. Technical report, ESA, April 1993.

[2] R. Carli, R. Timm. The Broadcast Packet. Technical Note on Content and Use of the BCPKT. ESA, 5.March 1998 INT–TN–18319.

[3] F.Ciceri. IBIS On–Board Data Management User Requirements Document. Technical Report 2, LABEN, Januar 1998. IN–IB–lAB–UR–0001.

[4] Decker, Oliver. Entwicklung einer Software Simulation des Hardware Event Pre–Processors des Integral Satelliten. Diplomarbeit. Universität Tübingen, Institut für Astronomie und Astrophysik, 1998.

[5] A. K. Dewdney. Computer–Recreations. In the game called Core War hostile programs engage in a battle of bits. In: Scientific American, May 1984, page 15.

[6] G. Drosdowski. *Duden Grammatik der deutschen Gegenwartssprache*, Volume 4. Dudenverlag, Mannheim, Leipzig, Wien, Zürich. 1984.

[7] ESA. INTEGRAL Experiment Interface Document, Part A for Instruments, September 1998. EID Part A.

[8] N. Gehrels and C. Winkler. International Gamma–Ray Astrophysics Laboratory (INTEGRAL): A future ESA mission for gamma–ray astronomy. In Ramsey and Parnell [19], pages 210–216.

[9] R. Giacconi, H. Gursky. *X-Ray Astronomy.* Volume 43. D.Reidel Publishing Company. Dordrecht–Holland/Boston–U.S.A. 1974.

[10] H. Herold, M. Meyer. *SCCS und RCS. Versionsverwaltung unter UNIX.* Volume 6. Addison–Wesley. 1995.

[11] IBIS Team. INTEGRAL Experiment Interface Document, Part B for Instruments, September 1998. IN.IB.IASW.EID.0022/9/98.

[12] E. Klatt. *Langenscheidt's Standard Dictionary of the English* and German Lanuages. Hodder and Stoughton, 1974.

[13] Nikolai von Krusenstiern. Entwicklung und Realisierung des Hardware Event Processors für den Imager IBIS an Bord des Satelliten INTEGRAL.

Diplomarbeit. Universität Tübingen, Institut für Astronomie und Astrophysik, 1998.

[14] C. Labanti et al. PICsiT: The high energy detection plane of the IBIS instrument onboard INTEGRAL. In Ramsey and Parnell [19], pages 269–279.

[15] F.Lebrun et al. ISGRI: A CdTe array imager for INTEGRAL. In Ramsey and Parnell [19], pages 258–268.

[16] H. L. Ledgard. *ADA. An Introduction; ADA Reference Manual.* Springer Verlag. New York, Heidelberg, Berlin, July 1980.

[17] G. G. Lichti et al. The spectrometer SPI of the INTEGRAL mission. In Ramsey and Parnell [19], pages 217–233.

[18] Manfred Nagel. *ADA*. Vieweg, 3 edition, 1991

[19] Brian D. Ramsey and Thomas A. Parnell, editors. *Gamma Ray and Cosmic-Ray Detectors, Techniques, and Missions*, volume 2806. The International Society for Optical Engineering, SPIE, August 1996.

[20] M. Rennie. Software Interface Control Document, INTEGRAL DPE CSSW. Technical Report, GMV, January 1999. INT–IC–GMV–0001.

[21] M. Rennie. Software Requirements Document, INTEGRAL DPE CSSW. Technical Report, GMV, July 1998. INT–RS–GMV–0001.

[22] L. Sarlo, M. Sella. Integral Packet Structure Definition. Technical Report, Alenia, February 1998. INT-RP-AI-0030.

[23] Diana Schmidt. *Erfolgreich Programmieren mit ADA.* Springer–Verlag, 1994.

[24] H. W. Schnopper et al. Joint European X–ray Monitor (JEM–X) for ESA's INTEGRAL mission. In Ramsey and Parnell [19], pages 297–307.

[25] H. J. Siegert, U. Baumgarten (Technische Universität München). *Betriebssysteme, Eine Einführung*, R. Oldenbourg Verlag, München, Wien, 1998.

[26] Defense Advanced Research Projects Agency, Department of Defense: *Requirements for a Higher Order Computer Programming Languages — ≫STEELMAN≪*. Arlington, VI 1978

[27] Martin Stuhlinger. Die Echtzeit–Datenverarbeitung des IBIS–Experiments auf INTEGRAL. Diplomarbeit. Universität Tübingen, Institut für Astronomie und Astrophysik, 1999.

[28] A. S. Tannenbaum. *Betriebssysteme, Entwurf und Realisierung*, Teil 1. Carl Hanser Verlag. München, Wien, 1990.

[29] TLD Systems. Reference Document for the TLD ADA Compiler, MIL–STD–1750A Target. Technical Report, TLD, March 1996.

[30] TLD Systems. Reference Document for the TLD Run Time System, MIL–STD–1750A Target. Technical Report, TLD, March 1996.

[31] TLD Systems. Reference Document for the TLD Extended Memory Linker, MIL–STD–1750A Target. Technical Report, TLD, March 1996.

[32] P. Ubertini et al. The Imager on Board INTEGRAL. In Ramsey and Parnell [19], pages 246–257.

[33] Reiner Volkmer. IBIS Communication Protocol Definition. Technical Report, IAAT, Juli 1998. IN-IM-TUB-ICD-01.

[34] Reiner Volkmer. HEPI Interface Description. Technical Report, IAAT, December 1998. IN-IM-TUB-TN/EL-018.

[35] Reiner Volkmer. HEPI Design Report. Technical Report, IAAT, December 1998. IN-IM-TUB-DES-001.

[36] Reiner Volkmer, Eckart Göhler, Patrick Risse and Martin Stuhlinger. Software Design Document, 5.0 edition, Technical Report, IAAT. Januar 1998. IN-IM-TUB-SDD-001.

[37] A. Robert Weiß. IBIS Detector Simulator, Detsim V.1.0. Technical Report, IAAT, Juni 1998.

Index