SuperHERO 大気球実験に向けた硬 X 線撮像分光用 CdTe 検出器データ読み出しシステムの開発

(Development of a Data Acquisition System for the CdTe Hard X-ray Imaging Spectrometer in the SuperHERO Balloon Mission)

東京大学大学院
理学系研究科 物理学専攻
馬場研究室

藤本 源

令和 8 年 1 月 21 日

概 要

X 線天文学において、 Chandra 衛星や XMM-Newton 衛星などの軟 X 線帯域（0.1–10 keV）での観測は高い空間分解能とエネルギー分解能を実現し、宇宙の高エネルギー現象の理解に大きく貢献してきた。一方で、硬 X 線帯域（10–80 keV）において NuSTAR 衛星や Hitomi 衛星などが観測を行い、Chandra 衛星や XMM-Newton 衛星では捉えられない超新星残骸や PWN などの天体が放出する非熱的放射の観測に成功している。しかし、これらの NuSTAR 衛星や Hitomi 衛星は、軟 X 線帯域に比べて空間分解能やエネルギー分解能が劣る。超新星残骸のショック面の高磁場環境下での細いフィラメント構造や、短時間で変動する「knot」と呼ばれる小規模構造の観測には、NuSTAR 衛星や Hitomi 衛星の分解能よりも高い分解能が必要である。

SuperHERO 計画は、硬 X 線帯域において高い空間分解能とエネルギー分解能を実現することを目的とした計画であり、2028 年の秋に大気球実験による実証実験を予定しているプロジェクトである。SuperHERO では東京大学が開発した CdTe-DSD(CdTe 両面ストリップ検出器を用いた硬 X 線撮像分光器) と、NASA/MSFC が開発した NiCo ミラーを組み合わせることで、硬 X 線帯域においても高い空間分解能とエネルギー分解能を実現することを目指している。しかし、 CdTe-DSD と NiCo ミラーが存在しているだけでは、 SuperHERO の目的を達成することはできない。CdTe-DSD からのデータを適切に読み出し、処理し、保存するためのデータ読み出しシステムが必要であり、それはまだ未完成だった。

本研究ではこの SuperHERO に必要である CdTe-DSD 用データ読み出しシステム (DE) の開発を一から行ない完成させた。SuperHERO には 7 台の CdTe-DSD が搭載される予定であり、各 CdTe-DSD は独立に動作をして、データを FPGA のバッファーに保存する。DE はこの独立して動作する 7 台の CdTe-DSD からデータを収集し、それを上流にまとめて送信する役割を担う。特に、 DE と DE の上流のシステムで SuperHERO のメインコンピュータである DEU 間の通信プロトコルやデータフォーマット自体の設計も本研究で行い、高負荷な状況や、異常な状態においても安定して動作することを目指した DE の開発を行った。それと同時に上流のシステムの開発に不可欠である検出器シミュレーターも開発した。この検出器シミュレーターは検出器の動作を正確にシミュレートし、DE のソフトウェアと繋げることで、実観測のルーチンを事前に検証可能にした。

最後に、これらの開発した DE の性能評価を行い、DE が設計通りに動作するか検証し、DE が可能とするデータ収集速度の上限を求めた。特に 15 時間にわたる長時間連続動作試験を行い、DE が SuperHERO の大気球実験に耐えうる信頼性を持つことを実証した。

目 次

第 1 章　序章 **1**
1.1　X 線天文学 1
1.2　硬 X 線天文学 1
1.2.1　銀河放射線の起源と超新星残骸 2
1.3　硬 X 線での高解像度撮像の重要性 3

第 2 章　SuperHERO 計画 **5**
2.1　SuperHERO 計画の概要 5
2.2　2028 年気球実験計画の概要 5
2.3　観測装置の概要 7
2.4　DUA と DE の構成 9
2.4.1　SpaceWire 通信プロトコル 9
2.4.2　DUAs + DE の構成図 9
2.4.3　SPMU001 10
2.5　CdTe-DSD 検出器 12
2.5.1　CdTe-DSD の読み出し ASIC 14
2.5.2　CdTe-DSD の ASIC から出力されるデータフォーマット 17
2.6　データフォーマット 18
2.6.1　DRAM に記録されるデータフォーマット 18
2.6.2　HK データ 20
2.6.3　タイムコードテーブル 20
2.6.4　HK データのフォーマット 20

第 3 章　CdTe 検出器の読み出しシステム (DE) の開発 **23**
3.1　読み出しシステム (DE) の役割と要求事項 23
3.2　DE / DEU 間通信インターフェースの規定 26
3.2.1　gRPC プロトコルの概要 26
3.2.2　High Level / Low Level コマンド 27
3.2.3　データ転送 29
3.3　DE のソフトウェアの構成 30
3.3.1　DE のクラス図と主要コンポーネント 30
3.3.2　DE のモード遷移 32
3.3.3　要求事項を満たすデータ収集方法の確立 33
3.3.4　DataRecorder 34

3.3.5 HKRecorder . . . 35
3.3.6 Scheduler . . . 35
3.4 DE を使用した観測の流れ . . . 38
3.5 まとめ . . . 40

第 4 章 DUA シミュレータの作成 41
4.1 DUA シミュレーターの必要性 . . . 41
4.2 DUA シミュレーターの構成 . . . 42
4.3 模擬データの生成方法 . . . 43
4.4 実際の運用 . . . 45
4.5 まとめ . . . 47

第 5 章 DE の性能評価 48
5.1 はじめに . . . 48
5.2 DUA の性能評価の重要性 . . . 48
5.3 DUA のハードウェア限界性能の評価 . . . 49
5.3.1 評価方法 . . . 49
5.3.2 SpaceWire のリンクレートとデータレートの関係 . . . 50
5.3.3 セットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係 . . . 50
5.3.4 小データの読み出しにかかる時間の評価 . . . 52
5.3.5 1 イベントフレーム (32KB) の読み出しにかかる時間の評価 . . . 54
5.3.6 まとめ . . . 56
5.4 DE の性能評価 . . . 57
5.4.1 実験装置 . . . 57
5.4.2 実験方法 . . . 58
5.4.3 実験結果 . . . 58
5.4.4 CdTe-DSD のデッドタイム実測と最大イベントレート . . . 60
5.4.5 DE の最大処理能力と CdTe-DSD 最大イベントレートの比較 . . . 60
5.5 まとめ . . . 62

第 6 章 Si-DSD を用いた長時間動作試験 63
6.1 長時間動作試験の必要性 . . . 63
6.2 実験目的 . . . 63
6.3 実験方法/実験装置 . . . 64
6.4 実験結果 . . . 69
6.4.1 取得データの健全性 . . . 69
6.4.2 長期安定性の検証結果 . . . 71
6.5 まとめ . . . 73

第 7 章 結論 74

付 録A SpaceWire のルーティング **77**
A.1 Path Addressing 77
A.2 Logical Addressing 78
A.3 SuperHERO での Addressing 78

付 録B 読み出し系システムを開発するにあたっての設計思想 **79**

付 録C コンパイル安全性と実行時安全性 **81**

付 録D spw_rmap の開発 **83**
D.1 背景と設計思想 83
D.2 spw_rmap の使用例 C++ 85
D.3 spw_rmap の使用例 Python 86

付 録E Nix を使用した DE **87**

付 録F DE を使用した地上試験用プログラムの開発 **88**

付 録G 取得データの解析プログラムの開発 **90**

付 録H エネルギー較正 **91**

付 録I 開発にあたって初期設計から修正を要した課題 **93**
I.1 High Level コマンドの設計 93
I.2 TimeCode とスケジューリング 94
I.3 Raspberry Pi 4 と PC との接続方法と IP アドレス管理 94
I.4 高負荷時における Read データ破損問題 95

図 目 次

1.1 宇宙線のスペクトルと「Knee」の位置付け (出典 [1]) 2
1.2 Chandra 衛星および NuSTAR 衛星による Crab Nebula 観測画像の比較 . 4

2.1 SuperHERO の観測プラン (出典: [2]) 6
2.2 NuSTAR と SuperHERO による Crab PWN の観測シミュレーション結果 6
2.3 SuperHERO の全体構成図 (出典: [2]) 7
2.4 MMA の観測帯域とその有効面積 (出典: [2]) 8
2.5 SuperHERO での各ユニットの対応関係の模式図 8
2.6 7 台の DUA と 2 台の DE の接続図 10
2.7 Detector-FPGA の写真 11
2.8 SpW-FPGA と SpaceWire I/O ボードの写真 12
2.9 Si 結晶および CdTe 結晶における X 線光電吸収率のエネルギー依存性 (出典: xraydb [3] から計算) 13
2.10 CdTe-DSD の模式図 13
2.11 VA-TA の原理 (出典: VA64TA1 のデータシートより) 14
2.12 ADC の動作原理 (出典: VATA460 のデータシートより) 15
2.13 FPGA から ASIC へのレジスター設定の流れ 16
2.14 ASIC DATA のフォーマット 17
2.15 DRAM に記録されるデータのフォーマット (イベントフレーム) 19
2.16 TimeCode Table のフォーマット 20
2.17 TimeCode + 1PPS Table のフォーマット 21
2.18 HK データのフォーマット 22

3.1 DE のクラス図 31
3.2 DE のモード遷移図 32
3.3 DE を用いた観測の流れ 39

4.1 DE シミュレーターの構成 43
4.2 DE シミュレーターのプログラムの流れ 44
4.3 イベント間隔のヒストグラム 45
4.4 DUA シミュレータを使用して作成したイメージ 46

5.1 ハードウェア性能評価のセットアップ A 49
5.2 ハードウェア性能評価のセットアップ B 49

5.3 セットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係。 51
5.4 SPMU001 における 4 bytes の読み出しにかかる時間の分布 53
5.5 SPMU001 における 32,768 bytes の読み出しにかかる時間の分布 54
5.6 4 台の DUA と 1 台の DE の接続図 57
5.7 4 台の Detector-FPGA と 1 台の SpaceWire Router そしてそれに繋がれた DE (Raspberry Pi 4) の実験装置 57
5.8 4 台の Detector-FPGA からのデータを受信する DE の負荷試験結果 59

6.1 Si-DSD を用いた長時間動作試験の実験装置 64
6.2 Si-DSD を用いた長時間動作試験のセットアップ 65
6.3 恒温槽内部に配置された Si-DSD 65
6.4 恒温槽上部に設けられた窓 66
6.5 Si-DSD 上に配置された金属製のしおり 67
6.6 Si-DSD 検出器ユニットの内部 67
6.7 Si-DSD 検出器ユニットの外観 68
6.8 Si-DSD を用いた長時間動作試験で取得したスペクトル 69
6.9 Si-DSD を用いた長時間動作試験で取得したイメージ 70
6.10 Si-DSD を用いた長時間動作試験で取得したライトカーブ 71
6.11 Si-DSD を用いた長時間動作試験で取得したイベント間隔のヒストグラム . 72

A.1 各ノードと Router との関係図 77

D.1 spw_rmap を用いた RMAP 読み出しの例 (C++) 85
D.2 spw_rmap を用いた RMAP 読み出しの例 (Python) 86

F.1 DE を使用した地上試験用プログラムの Batch ファイル例 89

H.1 ASIC0 (左) と ASIC1 (右) 側のスペクトルとピークのフィッティング結果 91
H.2 ASIC0 (左) と ASIC1 (右) 側のエネルギー較正曲線 92

表 目 次

1.1 Chandra 衛星 と XMM-Newton 衛星 (MOS) の観測帯域および角度分解能の比較 4

3.1 DE 制御コマンド一覧 28

5.1 セットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係のフィッティング結果 52
5.2 32 768 bytes 読み出し時のリンクレート別速度比 55

H.1 ^{241}Am 線源照射時に観測される ^{237}Np 起源の輝線 ([4] より抜粋) 91

第1章　序章

本章では、 X 線天文学および硬 X 線天文学が高エネルギー天体物理の理解において果たしてきた役割について述べる。まず 1.1 節では、 X 線天文学の成立背景と観測手法を示し、X 線天文学が天体物理学に重要な役割を果たしていることを示す。続いて 1.2 節、1.2.1 節および 1.3 節 では、10 keV 以上の硬 X 線帯域に着目し、非熱的放射の観測が超新星残骸における粒子加速現象や銀河放射線の起源を理解する上で重要であることを示す。最後に NuSTAR や Hitomi/HXI によって硬 X 線撮像観測は実現したものの、空間分解能には依然として制約が残されており、硬 X 線帯域においても数秒角級の高い角度分解能が求められていることを説明する。

1.1　X 線天文学

X 線天文学は高エネルギーの天体物理現象を研究する上で不可欠な観測手法である。宇宙に存在する大部分のバリオン物質は $10^5 - 10^{17}$K の高温状態にあることが予測されており [5]、この高温状態のプラズマを観測するためには EUV 観測や X 線観測が不可欠なためである。しかし、地球の大気は X 線をほぼ完全に吸収してしまうため、X 線天文学を行うためには、人工衛星や高高度気球などの大気圏外からの観測が必要であった。

XMM-Newton 衛星や Chandra 衛星はその高い軟 X 線帯域における高精度観測により、超新星残骸における粒子加速現象や星間物質との相互作用の研究に大きく貢献してきた。特に Chandra による高分解能の撮像・分光観測は、超新星残骸内部における粒子加速領域の微細構造や元素分布を詳細に描き出すとともに、衝撃波が周囲の星間物質（ISM）と相互作用する様子を直接的に捉えることを可能にした [6]。これにより、ISM の密度不均一性が衝撃波の形状や膨張速度に与える影響や、衝撃波前後における非熱的放射の起源に関する理解が大きく進展した。

1.2　硬 X 線天文学

主に 10 keV 以上のエネルギー帯で観測される硬 X 線は、高エネルギー天体物理現象を研究する上で重要な波長帯である。特に 10 keV 以下の軟 X 線帯域において支配的となる熱的放射の寄与を受けにくく、非熱的放射成分を比較的直接的に観測できるためである。パルサー風星雲、超新星残骸、ならびに活動銀河核における相対論的ジェットなどの天体現象では、電子が相対論的エネルギーまで加速され、シンクロトロン放射や逆コンプトン散乱といった非熱的放射を放出することが知られている。特に超新星残骸においては、衝撃波近傍で電子が TeV 級以上のエネルギーまで加速されることが示されており、これに

起因する非熱的放射は 10 keV 以上の硬 X 線帯域で顕著に観測される。実際、SN1006 に代表される一部の超新星残骸では、X 線放射の大部分が高エネルギー電子によるシンクロトロン放射によって支配されていることが報告されている [7]。

XMM-Newton 衛星や Chandra 衛星などの X 線天文衛星は X 線望遠鏡のミラーの性能により、10 keV 以下の軟 X 線帯域での観測に限定されてきたが、多層膜ミラーの発展により硬 X 線を観測することが可能な衛星が発展してきた。NuSTAR や Hitomi/HXI などがその例である。NuSTAR は 3-78.4 keV の帯域での観測を可能とし、空間分解能は 58 秒角 (HDP) を持っている。同様に Hitomi 衛星も 5-80 keV の帯域での観測を可能とし、空間分解能は 1.7 分角 (HDP) を持っている。しかし、これらの衛星は、既存の X 線衛星では成し得なかった硬 X 線帯域での観測を可能としたが、空間分解能に関しては、従来の軟 X 線衛星と比較するとまだまだ劣っている。

1.2.1 銀河放射線の起源と超新星残骸

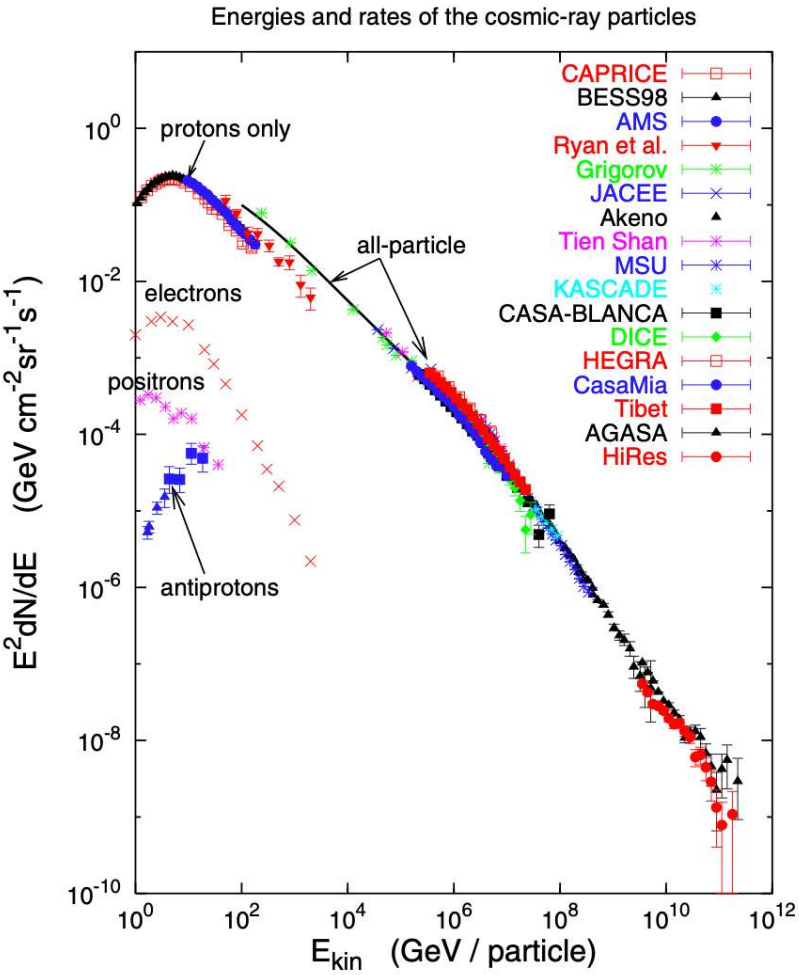


図 1.1: 宇宙線のスペクトルと「Knee」の位置付け (出典 [1])

図 1.1 は、宇宙線の全粒子エネルギースペクトルを示している。ここに示されるように、

宇宙線のエネルギースペクトルは概ねべき乗則に従うことが知られているが、$10^{15.5}$eV 付近において「knee」と呼ばれる特徴的な折れ曲がり構造を示す。このエネルギー領域を境としてスペクトルの傾きは急激に変化し、宇宙線の起源や加速機構の違いを反映している可能性が指摘されている。一般に、knee 以下のエネルギー帯に属する宇宙線は銀河放射線と呼ばれ、その主な起源は銀河系内に存在すると考えられている。

銀河放射線の有力な加速源としては、超新星残骸（SNR）が挙げられる。特に、 SNR の衝撃波において生じる第一種フェルミ加速と呼ばれる統計的粒子加速過程は、観測される銀河宇宙線のエネルギースペクトルを自然に説明し得る理論的枠組みを与えるものと考えられている [8]。一方で、超新星残骸が銀河放射線の主要な起源であることを直接的に裏付ける観測的証拠は依然として限定的であり、その検証は高エネルギー宇宙線観測および高感度放射観測における重要な課題の一つとなっている。

1.3 硬 X 線での高解像度撮像の重要性

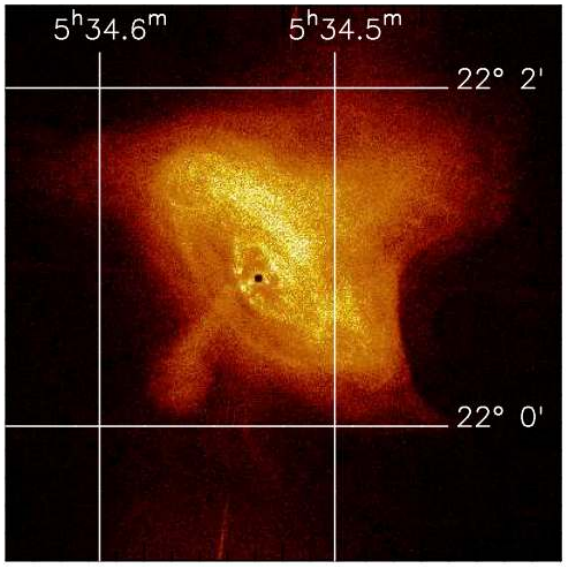
SNR や PWN における加速現場の詳細な理解には、硬 X 線帯域における高角度分解能撮像観測が不可欠である。硬 X 線帯域において高い角度分解能を備えた観測が不可欠である理由の一つとして、SNR の衝撃波近傍や、PWN の終端衝撃近傍では、高エネルギー電子が磁場中を運動することでシンクロトロン放射を放出し、細かな空間構造を形成することが挙げられる。特に、SNR 外縁に形成される薄いフィラメント構造や、近年の X 線観測により確認されている「knot」と呼ばれる短時間スケールで変動する局所的な高輝度構造 [9] は、粒子加速効率やエネルギー損失時間スケールを制限する上で重要である理由の一つである。これらの微細構造を直接的かつ定量的に理解するためには、硬 X 線帯域において既存の硬 X 線観測装置よりも高い角度分解能を備えた観測が不可欠である。

さらに電波帯域との高精細な比較研究を行うためにも、高角度分解能の硬 X 線撮像観測が必要である。ALMA に代表されるミリ波・サブミリ波干渉計観測では、分子雲内部の密度揺らぎや衝撃波起源と考えられる微細構造が、約 5 秒角程度の空間スケールで分解することが可能である。例えば Orion A 分子雲を対象とした観測研究でも、ALMA の高空間分解能を用いてフィラメント構造が詳細に解析されている [10]。SNR における粒子加速領域と、電波帯域で観測される分子雲構造との対応関係を詳細に検証するためには、ALMA と同等の角度分解能を持つ硬 X 線での 5 秒角程度の高い角度分解能での撮像観測が必要である。

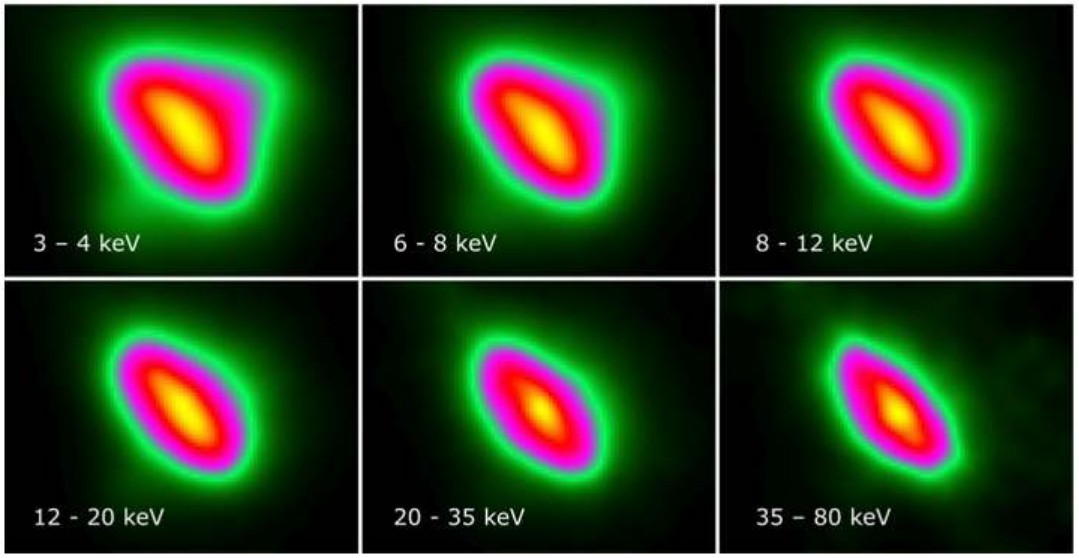
表 1.1 に示すように、Chandra 衛星や XMM-Newton 衛星は軟 X 線帯域において高い角度分解能を実現している一方で、NuSTAR 衛星や Hitomi 衛星搭載の HXI は硬 X 線帯域において分角程度の角度分解能に留まっている。 図 1.2 は Chandra 衛星と NuSTAR 衛星による Crab Nebula の観測画像の比較を示している。Chandra 衛星による Crab Nebula の観測画像は非常に高い空間分解能を持ち、詳細な構造を捉えているのに対し、NuSTAR 衛星による同天体の観測画像は、空間分解能が劣るために詳細な構造を捉えることができていない。

表 1.1: Chandra 衛星 と XMM-Newton 衛星 (MOS) の観測帯域および角度分解能の比較

観測機器	観測帯域 (keV)	FWHM (arcsec)	HEW (arcsec)
Chandra (HRMA)	0.1–10	< 0.5	< 1
XMM-Newton MOS	0.15–12	4.5–6.0	12–14
NuSTAR	3–78.4	~ 58	
Hitomi / HXI	5–80	~ 102	



(a) Chandra 衛星による Crab Nebula の観測画像（出典：[11]）



(b) NuSTAR 衛星による Crab Nebula の観測画像（出典：[12]）

図 1.2: Chandra 衛星および NuSTAR 衛星による Crab Nebula 観測画像の比較

第2章　SuperHERO 計画

本章では、SuperHERO 計画の目的および観測装置の概要について示す。まず 2.1 節では、SuperHERO の計画概要について述べる。SuperHERO 計画では NiCo ミラーと CdTe-DSD 検出器を組み合わせた焦点距離 12 m の光学系により、硬 X 線帯域における高精細撮像を目指す。続いて 2.2 節および 2.5 節では、SuperHERO のハードウェア構成について述べる。2.4 節以降は、3 章で開発を行った DE の設計を理解するために必要な DUA と DE の構成について説明を行う。特に 2.5 節では SuperHERO で使用される CdTe-DSD 検出器とそのデータフォーマットについて説明を行い、3 章でのデータ処理内容を理解するために必要な基礎知識について述べる。

2.1　SuperHERO 計画の概要

SuperHERO 計画は、NASA/MSFC(マーシャル宇宙飛行センター) および東京大学を中心とする国際的な研究機関が連携して進めている、高分解能硬 X 線望遠鏡の開発計画である。本計画は、従来の硬 X 線観測では困難であった微細構造の撮像を可能にすることを目的とし、成層圏気球を用いた実証実験を通じて新たな観測技術の確立を目指している。現在は、2028 年秋にアメリカで予定されている気球実験に向けて装置開発および性能評価が進められており、この実験では約 10 秒角の角度分解能の達成が主要な技術目標とされている。さらに SuperHERO 計画は、将来的な長時間気球観測や宇宙望遠鏡ミッションへの発展を視野に入れている。最終的には、NuSTAR を大きく凌駕する約 5 秒角級の角度分解能を持つ次世代硬 X 線望遠鏡の実現を目標とし、高エネルギー天体物理学における新たな観測基盤を構築することを目的としている。

2.2　2028 年気球実験計画の概要

2028 年の観測では、 SuperHERO で使用される観測装置が 10 秒以下の角度分解能を達成できるかどうかを実証することを目的としている。気球は 2028 年秋にアメリカのニューメキシコ州 Fort Sumner から打ち上げられ、約 2 日間の観測を行う予定である。観測では高度約 39.5 km の成層圏に達し、20 keV 以上の硬 X 線の大気吸収が非常に小さい領域で観測を行う。観測の一次ターゲットは、かに星雲 (Crab PWN) であり、パルサー風終端衝撃波での、シンクロトロン放射からの硬 X 線を撮像観測することを目的としている。

図 2.1 に SuperHERO の観測プランを示す。この観測プランに示されているように、Crab PWN は夕方から夜間にかけて約 8 時間観測される予定である。観測レートは約 2.7

counts/s (20–40 keV) と見積もられており、約 8 時間の観測時間中に約 70,000 counts のイベントが検出されると予想されている。もし Crab PWN が観測できない時間帯には、Hercules X-1 を点源ターゲットとして観測を行い、SuperHERO の角度分解能を実証する予定である。Hercules X-1 の観測時間は 図 2.1 に示されているように約半日間である。

図 2.2 (出典 [2]) に NuSTAR と SuperHERO による Crab PWN の観測シミュレーション結果を示す。左側が NuSTAR による観測結果であり、中央が 10 秒角の角度分解能を持つ SuperHERO による観測結果である。最後の右側が 5 秒角の角度分解能を持つ SuperHERO による観測結果である。2028 年の気球実験では 10 秒角の角度分解能を目指しており、中央のシミュレーション結果のような高精細な撮像観測が期待されている。

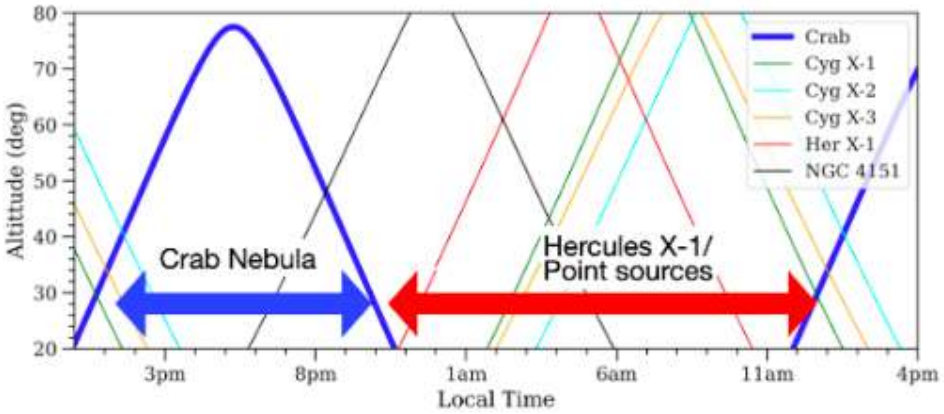


図 2.1: SuperHERO の観測プラン (出典: [2])

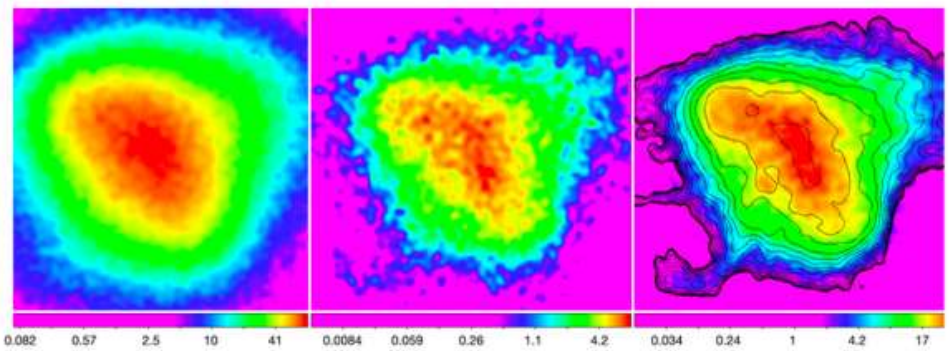


図 2.2: NuSTAR と SuperHERO による Crab PWN の観測シミュレーション結果 (出典: [2])。左が NuSTAR による 20–40 keV での観測結果、中央が 10 秒角の角度分解能を持つ SuperHERO による観測結果のシミュレーション、右が 5 秒角の角度分解能を持つ SuperHERO による観測結果のシミュレーション。

2.3　観測装置の概要

本実験計画では、図2.3 に示すように、7 基の NiCo 多層膜ミラーを用いた光学系（Mirror Module Assemblies; MMAs）と 7 基の CdTe-DSD 検出器 [13] [14] [15] を用いた検出器系（Detector Unit Assemblies; DUAs）を搭載した観測装置が使用される。そして、これらの光学系と検出器系はトラスにより結合され、焦点距離 12 m の硬 X 線望遠鏡を構成する。

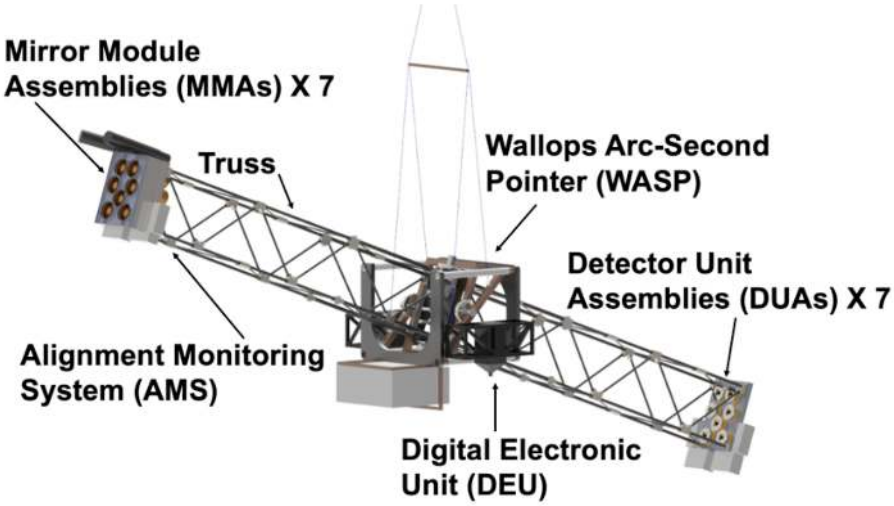


図 2.3: SuperHERO の全体構成図 (出典: [2])

MMA（Mirror Module Assembly）は NASA/MSFC により開発された、NiCo ミラーシェルを用いた硬 X 線望遠鏡用光学系である。SuperHERO では同一構成の MMA を 7 基用いて観測を行う。図 2.4 に、7 基すべての MMA を組み合わせた場合の観測エネルギー帯域と有効面積を示す。オレンジ色の曲線は単層イリジウムコーティングを施した場合の有効面積を、青色の曲線はタングステン／シリコン（W/Si）からなる深さ方向最適化多層膜コーティングを適用した場合の有効面積を表している。2028 年の気球実験では、各 MMA に 2 枚のミラーシェルが搭載され、単層イリジウムコーティングが用いられる。この構成における有効面積は図中のオレンジ色の曲線とほぼ同等であり、35 keV において約 45.5 cm^2 で最大となる。なお、本図には成層圏気球高度における大気吸収の影響は含まれておらず、20 keV 以下では大気による吸収が支配的となる。

DUA（Detector Unit Assembly） は、各望遠鏡の焦点面に配置される検出器であり、CdTe-DSD (CdTe Double-Sided Strip Detector) とその周辺回路を収めたユニットである。CdTe-DSD の信号は専用 ASIC (Application Specific Integrated Circuit) [16] と FPGA (Field Programmable Gate Array) により処理され内部バッファーに蓄積され、DUA から SpaceWire を介して後段の電子系へ送出される。1 台の DUA は 1 基の CdTe-DSD に対応し、SuperHERO では計 7 台が搭載される。

DE（Detector Electronics）は、検出器系に近接配置される検出器用電子系であり、複数 DUA からのデータを受け取り、集約・バッファリング・上位系への中継を行う中間段である。DUA–DE 間は SpaceWire により接続される。Linux 搭載の小型 CPU (Raspberry Pi 4) で構成される DE は受信データを蓄積し、Ethernet で上位の電子系へ送出する。

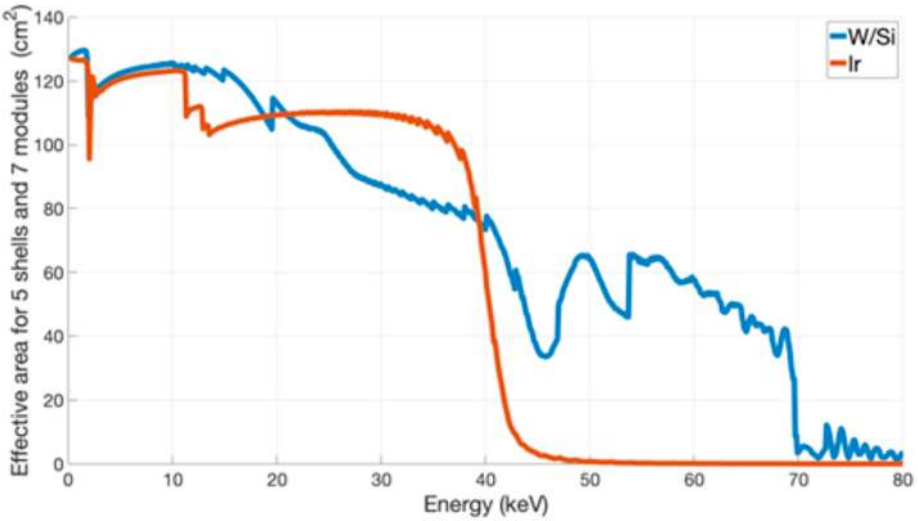


図 2.4: MMA の観測帯域とその有効面積 (出典: [2])

SuperHERO では 7 台の DUA に対し、2 台の DE が搭載される。

DEU（Digital Electronics Unit）は、観測装置全体を統括する上位電子系である。DEU は DUAs/DE からの科学データの記録、ハウスキーピングデータの収集、AMS (Alignment Monitoring System) の管理、電源生成および配電、ならびにヒータ制御などのシステム全体の統合制御を担う。DEU には様々な機能が搭載されているが、この研究では DEU は DE の上位に位置し、DE からのデータを受信・記録する外部 PC として扱う。

図 2.5 に SuperHERO での各ユニットの対応関係の模式図を示す。DUA は 7 台から構成され、それらで回収されたデータは 2 台の DE に送信される。その 2 台の DE は DEU にデータを送信する。すなわち観測されたデータは下流から DUA → DE → DEU の順に伝達される。CdTe-DSD を含む DUA とその間の SpaceWire 通信用 Router のハードウェアと FPGA のファームウェア、そして DE とそのソフトウェアは日本側が開発を担当している。一方、その上流に位置し、観測装置全体を統括する DEU は NASA 側が開発を担当している。

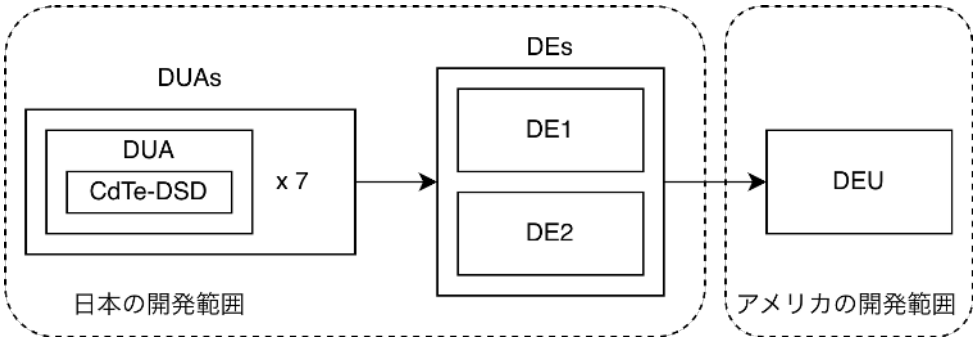


図 2.5: SuperHERO での各ユニットの対応関係の模式図

2.4 DUA と DE の構成

この節では 3 章の DE の開発と設計を理解するために必要な DUA と DE の構成について説明を行う。この節では日本側開発範囲である DUA と DE の構成と、DE の設計を理解するために必要な SpaceWire 通信プロトコルについて説明しながら、構成する各コンポーネントについて説明を行う。これらの構成は SuperHERO の開発段階である程度確定していたもであるが、DE のソフトウェアとして実際に動作可能であるか検証し、今回実際に 6 章で動作検証を行った構成である。

2.4.1 SpaceWire 通信プロトコル

SpaceWire は DUA の通信網を構成する通信プロトコルであり、ESA (European Space Agency) が開発した宇宙機器向けの高速な双方向通信プロトコルである [17]。SpaceWire の特徴は双方向に通信可能なシステムであり、大規模なネットワークを構築できることである。多くの宇宙機器で採用されているという実績があり、信頼性の高い通信プロトコルである。

この SpaceWire 通信プロトコルは DUA と DE 間には使用されている。特に DUA と SpaceWire Router 間では SpaceWire ケーブルを用いた物理的な SpaceWire 通信が行われており、DE と SpaceWire Router 間ではシマフジ電機の開発している SpaceWire-to-GigabitEther [18] 等で使用されている、TCP 上で動作する SpaceWire 通信 (SSDTP2) で行われている。

SpaceWire ではその上で動作するプロトコルとして RMAP (Remote Memory Access Protocol) が存在する。RMAP は SpaceWire ネットワーク上でリモートデバイスのメモリにアクセスするためのプロトコルであり、SuperHERO ではこの RMAP を用いて CdTe-DSD の FPGA のレジスタを操作し、データ収集と制御を行っている。RMAP ではメモリにアクセスする方法を複数提供しているが、Read と Write の根本的な操作に集約される。Read は特定のアドレスから特定のバイト数のデータを読み出す操作であり、Write は特定のアドレスに特定のバイト数のデータを書き込む操作である。複雑な操作として Read-Modify-Write も存在するが、これは Read と Write を同時に行う操作であり、複雑な操作が可能であるが word 単位での register の書き換え等が主であるため、SuperHERO では使用することはない。

2.4.2 DUAs + DE の構成図

DUA (Detector Unit Assembly) は CdTe-DSD とそれをコントロールする FPGA (SPMU001)、 それと CdTe-DSD に高電圧バイアスや電源を供給するユニットの総称である。しかし、DUA の電源周り等の電子機器関連はどのユニットが制御を担当するかは決定していない。つまり、 DUA 内部で Detector-FPGA が何を制御するかは明確に定義されていない。だが、CdTe-DSD の FPGA が制御をする場合であっても、 FPGA のレジスタを単に操作するものとなるため、今回の DE の設計には影響しない。そのため、こ

こでは 「CdTe-DSD と FPGA (SPMU001)」という、電源や高電圧バイアスのユニットを含まない最小構成で DUA を定義する。

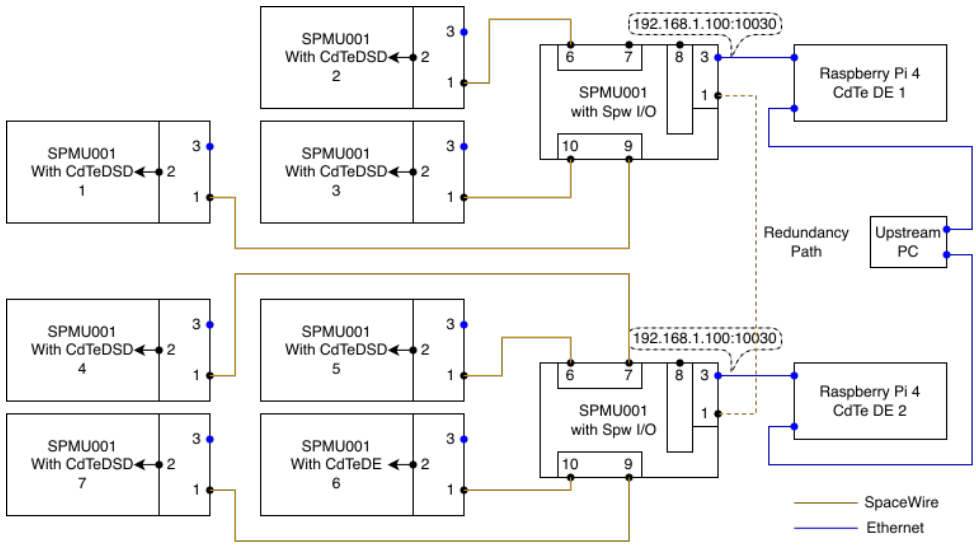


図 2.6: 7 台の DUA と 2 台の DE の接続図。図中の青線が SpaceWire ケーブル、黄色線が Ethernet ケーブルを示す。7 台の DUA は SpaceWire ケーブルを介して SpaceWire Router に接続されており、SpaceWire Router は SpaceWire to Ethernet ブリッジを介して DE に接続されている。

図 2.6 に 7 台の DUA と 2 台の DE の接続図を示す。各 DUA は SpaceWire ケーブルを介して SpaceWire Router に接続されており、SpaceWire Router は SpaceWire to Ethernet ブリッジを介して DE に接続されている。そして SpaceWire Router は冗長化のために相互に接続されており、DE からはどちらの Router にもアクセスできるようになっている。冗長パスは、片方の DE が故障したり、復帰不可能な状態に陥った場合に備えて設計されている。そのため、 DE は 3 台/4 台 または 7 台の DUA を制御する能力を有する必要がある。DE のソフトウェアではこの点を考慮して、任意の台数を制御可能なように設計されている。図 2.6 中に書き込まれている番号は SpaceWire の Routing (Appendix A) で使用される port 番号であり、SpaceWire 規格由来の識別番号である。

この構成上 3 台と 4 台という台数の違いは存在するが、 DE から見た DUA はシンメトリックであり、従って 2 つの DE 内で全く同じソフトウェアを動作させることとなる。特に、冗長パスは実運用上使用されないことが想定されるため、DE のソフトウェアでは DE 同士の通信は行われず、互いに干渉しあうことはない。また、図 2.6 に示すルーターの接続順番やポートの割り当ては任意であり、それぞれの CdTe-DSD が全て独立であるため、どのように接続しどのように配置するか柔軟に設計できる。

2.4.3 SPMU001

図 2.7 に SPMU001 の写真を示す。SPMU001 はシマフジ電機と IPMU が共同開発した汎用型物理計測用 FPGA ボードであり、AMD 製の FPGA (Spartan-7)、二つの SpaceWire

ポート、 DDR2 メモリを搭載している。SuperHERO で使用される DUA の FPGA ボードはこの SPMU001 を使用している。SPMU001 は主に CdTe-DSD を制御する FPGA と、 SpaceWire Router を制御する FPGA として同じものが使用されている。そのため、これ以降の説明では CdTe-DSD を制御する SPMU001 は Detector-FPGA、SpaceWire Router を制御する SPMU001 は SpW-FPGA と呼ぶこととする。そしてただ単に FPGA と呼ぶ場合は Detector-FPGA / SpW-FPGA の両方を指すこととする。

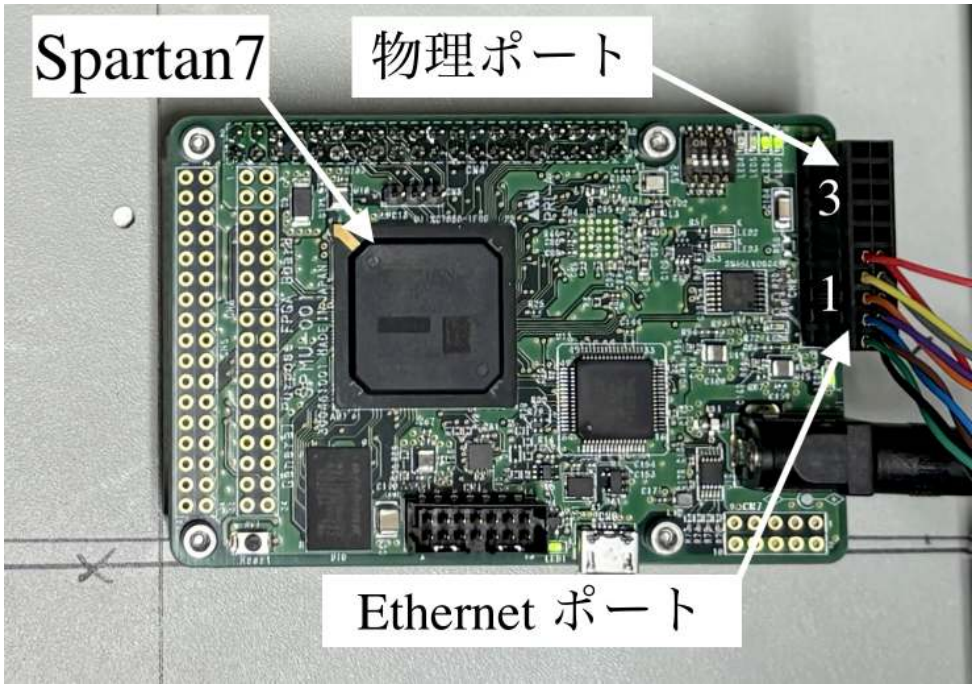


図 2.7: Detector-FPGA の写真

Detector-FPGA に搭載されている SpaceWire ポートには SpaceWire ケーブルを接続することができる、物理ポート一つ (port: 1) と TCP 上で SpaceWire 通信を行う SSDTP2 プロトコルで接続可能な Ethernet ポート (port: 2) 一つが存在する。この二つのポートを用いることで、 Detector-FPGA の仮想的な port (port: 2) にアクセスすることができ、Detector-FPGA 内部のレジスターや DDR2 メモリにアクセスすることができる。この DDR2 メモリは 128 MByte であり、その内の半分の 64 MByte が CdTe-DSD のリングバッファーとして使用されている。DE は定期的に Detector-FPGA のレジスターから write_pointer を読み出し、それを元に Detector-FPGA の DDR2 メモリからデータを読み出し、上流の DEU に送信する必要がある。

SpW-FPGA には SpaceWire を入出力するためのインターフェースボードとして SpaceWire I/O ボードが取り付けられている。図 2.8 に SpW-FPGA と SpaceWire I/O ボードの写真を示す。SpaceWire I/O ボード (左) は SPMU001(右) の拡張ボードとして動作し、SpaceWire の物理ポートを 5 ポート搭載している。SpW-FPGA 自体にも 1 ポートの物理 SpaceWire ポートが搭載されているため、SpW-FPGA と SpaceWire I/O ボードを組み合わせることで、合計 6 つの物理 SpaceWire ポートを提供することができる。SuperHERO では 7 台の DUA を制御する必要があるため、 4 台の DUA と 3 台の DUA をそれぞれ

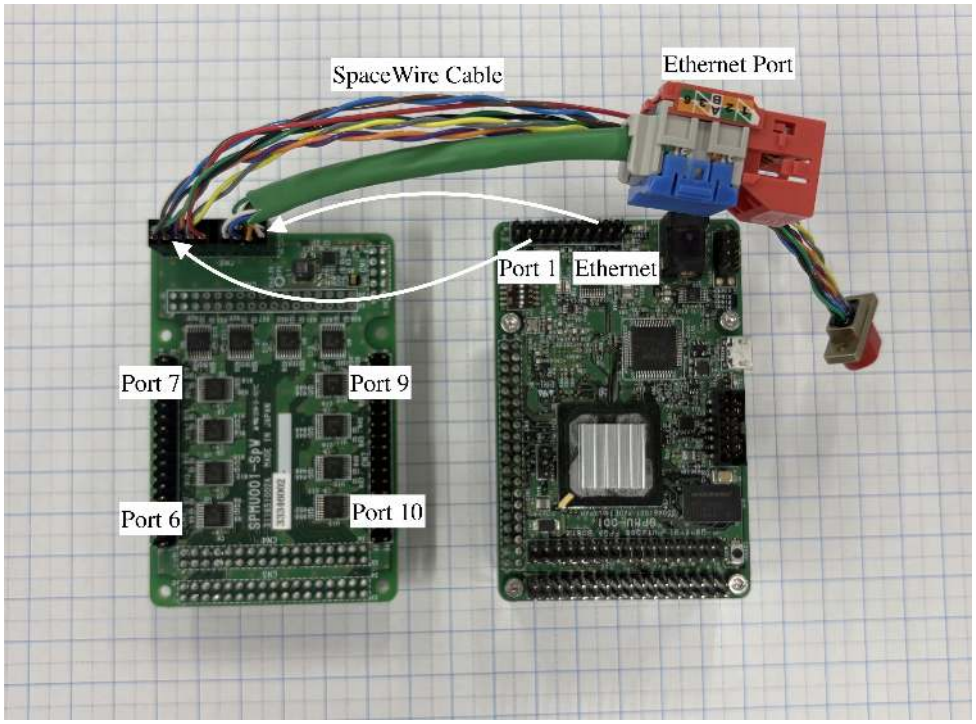


図 2.8: SpW-FPGA と SpaceWire I/O ボードの写真。 左側が SpaceWire I/O ボード、右側が SpW-FPGA である。左側の SpaceWire I/O ボードは、右側の SpW-FPGA のボードの上にソケットを介して取り付けることができる。

制御するために 2 台の SpW-FPGA + SpaceWire I/O ボードの組み合わせが使用される。以降はこの SpW-FPGA + SpaceWire I/O ボードの組み合わせを SpaceWire Router と呼ぶこととする。

2.5 CdTe-DSD 検出器

CdTe-DSD 検出器 [13] [14] [15] は SuperHERO で使用される検出器であり、従来のシリコン検出器と比べて硬 X 線で検出効率が高いことが特徴である。Hitomi 衛星 [19] に搭載された HXI で実証された技術を基に開発されており、高エネルギー X 線の検出に適した CdTe (テルル化カドミウム) を用いた両面ストリップ検出器である。Z=48 の Cd と Z=52 の Te から構成される CdTe は、高い原子番号に起因する高い光電吸収断面積を持つため、硬 X 線領域における検出効率が高く、図 2.9 は xraydb [3] を用いて Si 結晶と CdTe 結晶における X 線光電吸収率のエネルギー依存性を示したものである。この図からわかるように、 CdTe は Si と比較して高エネルギー X 線に対して非常に高い吸収率を持つ。例えば 80 keV では 1mm 厚の Si(Z=14) で吸収率が約 1% であるのに対し、CdTe では約 80% に達する。

図 2.10 に CdTe-DSD の模式図を示す。CdTe-DSD は 64 channel を持つ ASIC を片面に 2 枚ずつ、計 4 枚搭載しており、128 × 128 strips の両面ストリップ検出器である。CdTe-DSD の CdTe の結晶厚は 1mm であり、その両面にストリップ電極が形成されて

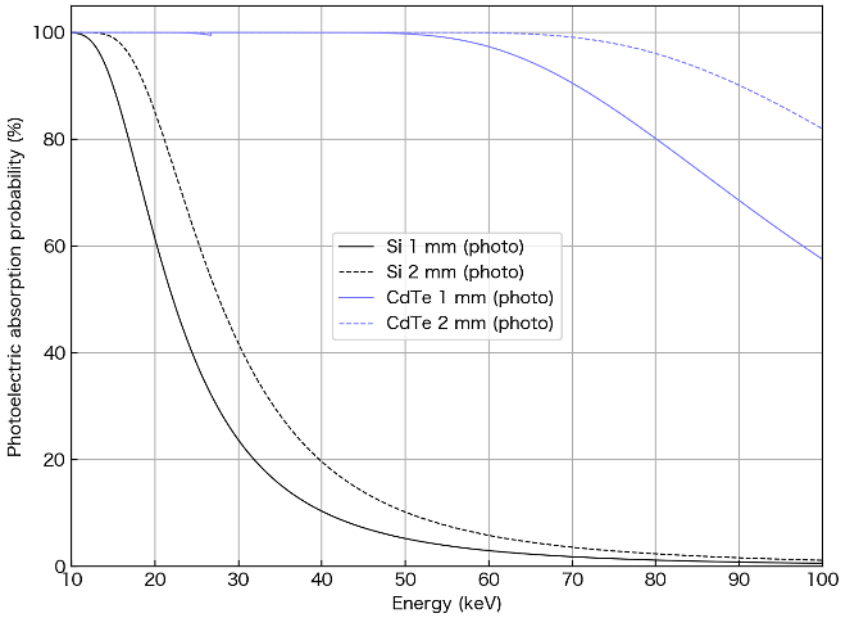


図 2.9: Si 結晶および CdTe 結晶における X 線光電吸収率のエネルギー依存性 (出典: xraydb [3] から計算)

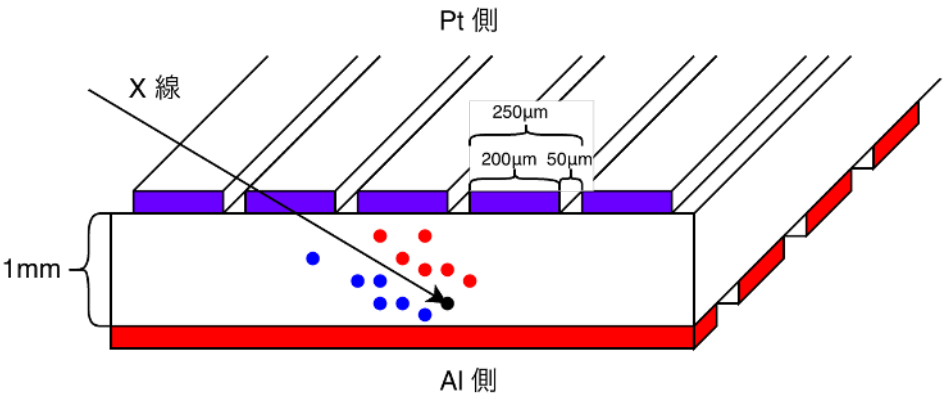


図 2.10: CdTe-DSD の模式図

いる。各ストリップのピッチは 250 μ m であり、各 strip は 50 μ m の間隔で配置されている。そのため、 CdTe-DSD の検出面積は 32mm × 32mm である。

X 線が検出器に入射すると、 CdTe 内で電子・正孔対が生成される。生成された電子・正孔対は電場によりストリップ電極に収集され、各ストリップに接続された専用 ASIC により信号が読み出される。両面ストリップ検出器の構造により、 X 線の入射位置を 2 次元的に特定することが可能である。ストリップ電極に回収された信号は専用 ASIC により増幅・整形され、ある閾値 (threshold) を超えた信号が検出イベントとして認識される。

2.5.1 CdTe-DSD の読み出し ASIC

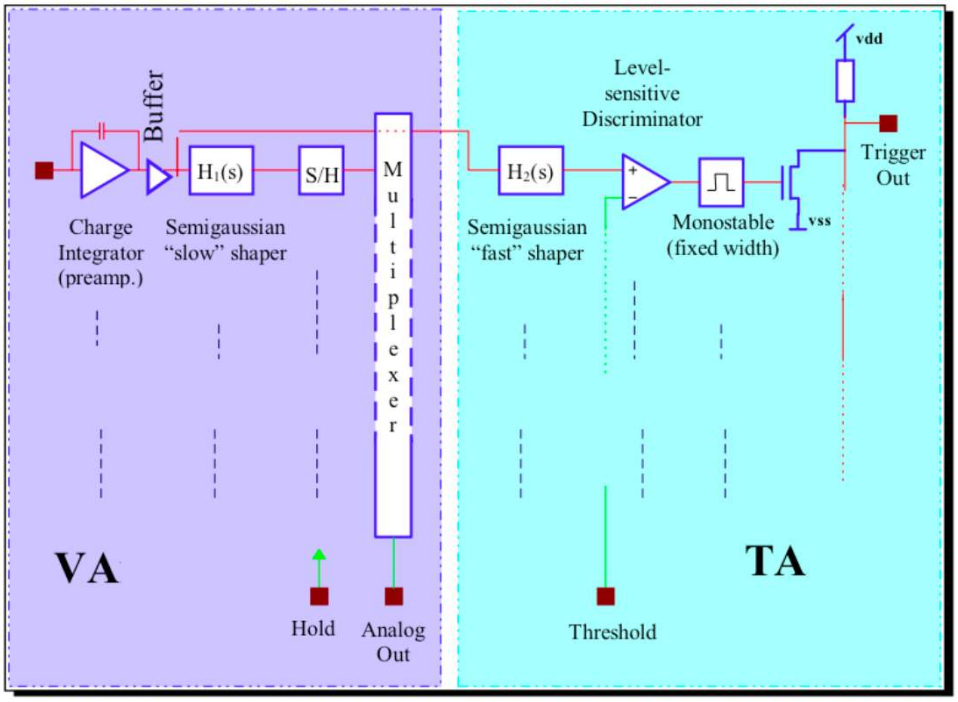


図 2.11: VA-TA の原理 (出典: VA64TA1 のデータシートより)

CdTe-DSD には専用の読み出し ASIC が搭載されており、この ASIC は CdTe-DSD の各ストリップからの信号を増幅・整形し、検出イベントを生成する。この ASIC は VATA450 と呼ばれるものであり、名前の通り VA 部分と TA 部分から構成されている。図 2.11 にこの VA-TA の原理を簡略化したブロック図を示す。VATA450 は 64 channel の同時読み出しが可能な ASIC であり、VATA450 は波高値の出力を行う VA 部分と、トリガー信号を生成する TA 部分から構成されている。CdTe から出力された信号はまず前段の電荷有感型増幅器 (Charge Sensitive Amplifier; CSA) で増幅され、VA と TA それぞれの整形増幅器に送られる。TA、 VA にはそれぞれ時定数が 600 ns 、 3–5 μs の整形増幅器が搭載されており、TA 部分に入力された信号が threshold を超えた場合にトリガー信号が生成される。この信号をもとに、 VA での整形時間に合わせた delay をかけ、整形された波形をサンプルホールドする。その後に shift_in_b 信号が入力されると読み出しシーケンスが開始され、クロック信号に合わせて 1 bit ずつデータが読み出される。

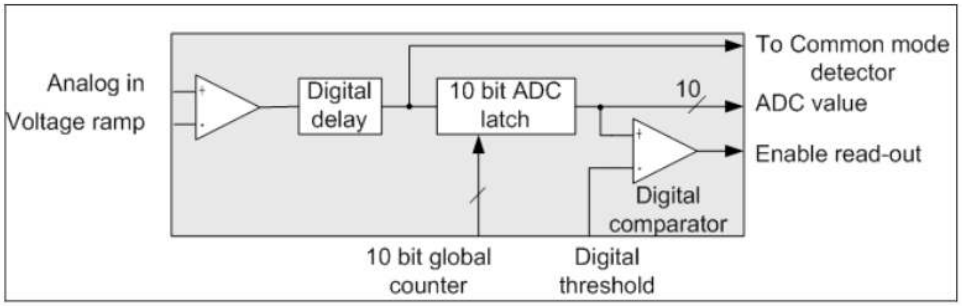


図 2.12: ADC の動作原理 (出典: VATA460 のデータシートより)

2.12 に各チャンネルの ADC の動作原理を示す。各チャンネルには 10 bit の ADC が搭載されており、ASIC 内部で各チャンネルごとにアナログ-デジタル変換が行われる。各チャンネルは Common Mode Detector に繋がる部分を持っている。channel の読み出しを決定する digital threshold は、ASIC のパラメータと Common Mode Detector の出力を加算器により加算した値として決定される。そしてその digital threshold を超えると Enable 信号が立ち、そのチャンネルの ADC が読み出される。この digital threshold を調整することで、 ASIC 全体のノイズレベルを調整することができる。

この ASIC は shift_out_b 信号があり、読み出しシーケンス終了時にこの信号が HIGH になることで読み出しシーケンスの終了を示す。この shift_out_b を次の ASIC の shift_in_b に接続することで、daisy chain 接続が可能であり、 1 本のシリアル信号で複数枚の ASIC からのデータを読み出すことができる。SuperHERO で使用される CdTe-DSD では 4 枚の ASIC が daisy chain 接続されており、1 本のシリアル信号で 4 枚の ASIC からのデータを読み出すことができる。

ASIC のレジスター設定も reg_out を次の ASIC の reg_in に接続することで daisy chain 接続が可能であり、 1 本のシリアル信号で複数枚の ASIC のレジスターを設定することができる。各 ASIC は 936 bit のレジスターを持っており、FPGA のレジスターには 128 bytes アラインメントされて 4096 bytes の固定長データとして格納されている。$936\text{bit} \times 4(\text{ASIC}) = 3744\text{bit}$ 分の bit データを一枚目の reg_in に接続されたシリアル信号で入力し、 load 信号を HIGH にすることで 4 枚の ASIC のレジスター設定を同時に行うことができる。また、 read_back 信号を HIGH にすることで、設定されたレジスターの内容をシフトレジスターに格納し、 reg_out からシリアルに出力することができる。このデータは FPGA のレジスターに 4096 bytes の固定長データとして格納される。

図 2.13 に FPGA から ASIC へのレジスター設定の流れを示す。ASIC のレジスター設定時には 0x44a21000 から始まる Detector-FPGA のレジスターに ASIC のレジスターを設定しそれを ASIC に送信し、 load 信号を HIGH にすることで ASIC のレジスター設定を行うことができる。その後、 read_back 信号を HIGH にして ASIC のシフトレジスターを読み出すと Detector-FPGA の 0x44a22000 から始まるレジスターに ASIC のレジスター設定内容が格納される。この内容を読み出し、 FPGA に設定した内容と比較することで ASIC のレジスター設定が正しく行われたか確認することができる。

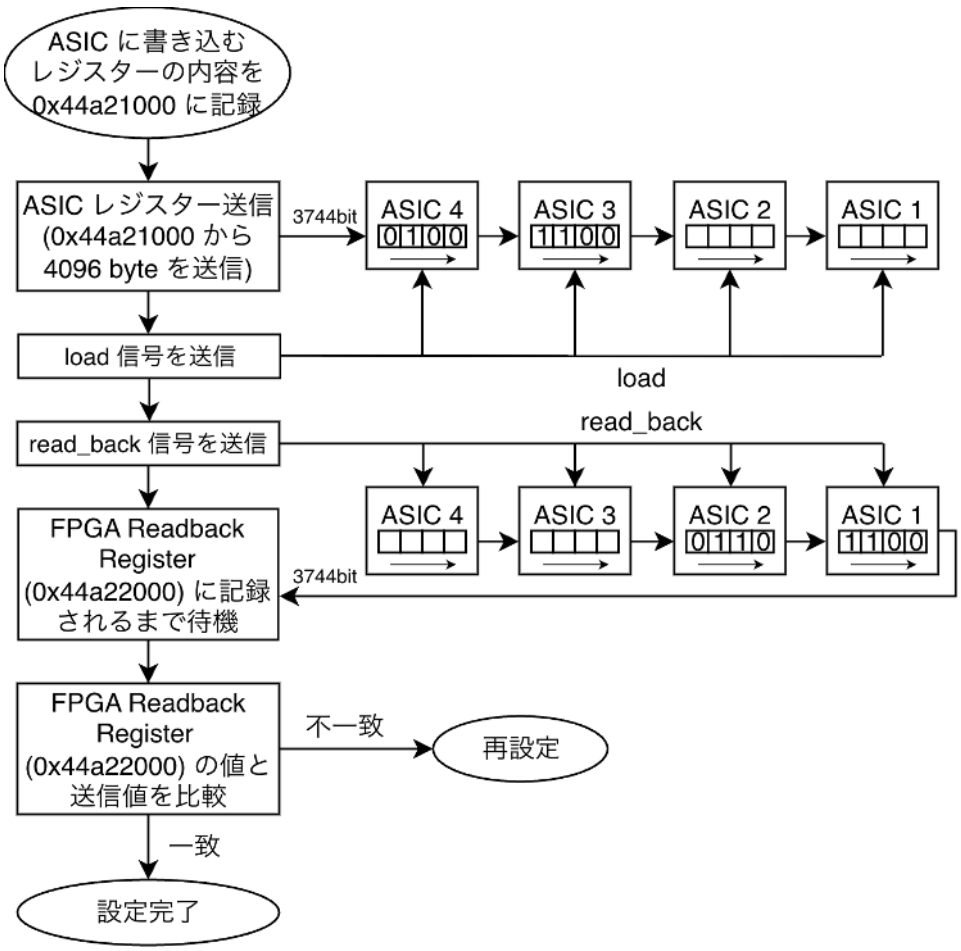


図 2.13: FPGA から ASIC へのレジスター設定の流れ

2.5.2 CdTe-DSD の ASIC から出力されるデータフォーマット

SuperHERO で使用されている、CdTe-DSD には 4 枚の ASIC が搭載されている。各 ASIC は 64 チャンネルの読み出しを可能としており、それぞれ Pt 側で 2 枚、Al 側で 2 枚の ASIC が搭載されている。したがって、ASIC 4 枚からなる合計 256 チャンネルの情報がフレームとして出力される。データは daisy chain された 4 枚のデータがシリアルに連結されて出力される。この ASIC から出されるフレームのフォーマットは図 2.14 に示す通りである。

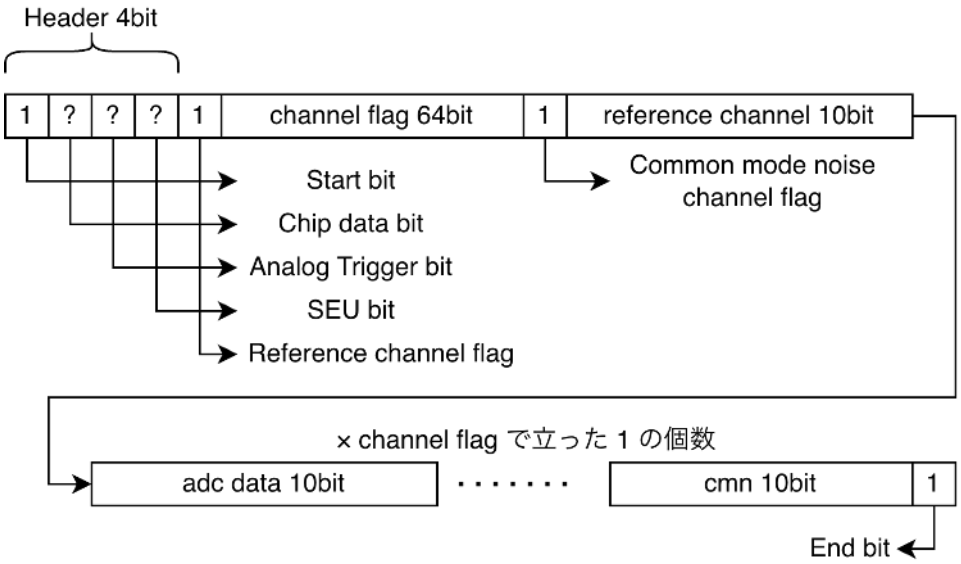


図 2.14: ASIC DATA のフォーマット

ASIC から出されるフレームは、 Pseudo Trigger であることを示す Chip Data ビットや、アナログトリガーからの信号であることを示す Analog Trigger ビット、SEU (single event upset) の発生を示す SEU Trigger ビットを含む 4 bit の Header から始まる。次に、1 bit の reference channel flag が続く、これは殆どの場合 1 である。続く 64 bit は Channel Flag であり、各チャンネルのデータが記録されているかを示すフラグである。次の 1bit は Common Mode Noise Flag であり、コモンモードノイズのデータが記録されているかを示すフラグである。この flag も殆どの場合 1 である。これらの Channel flag に続く形で、 10 bit の ADC データが続く、始めは、 1 チャンネル分のリファレンスチャネルのデータが 10 bit で続き、その後に N チャンネル分のイベントデータが 10 bit ずつ続く。N は 0 から 64 までの値を取り、 Channel Flag で立ったビットの数に対応する。その後に、コモンモードノイズのデータが 10 bit で続き、最後に End bit (1 bit) でフレームが終了する。つまり、 Trigger が発生したチャンネル数を N とすると、ASIC から出されるフレームのビット数は以下のように計算できる。

$$
\begin{aligned}
&5\ \text{(Header)} + 64\ \text{(Channel Flag)} + 1 \\
&+ 10\ \text{(reference channel)} + N \times 10\ \text{(Event data)} \\
&+ 10\ \text{(common mode noise)} + 1 \\
&= 10N + 91\ \text{bits}
\end{aligned}
\tag{2.1}
$$

この $10N + 91$ bits のデータが 4 bytes アラインメントされて SPMU 001 の DDR2 メモリに書き込まれる。そのため、全チャンネル読み出し (N=64) の場合 92 bytes のデータが一回の読み出しで生成される。

2.6 データフォーマット

ここでは、DE が扱う CdTe-DSD のデータフォーマットについて説明する。ASIC から出されるフレームフォーマット、及び DRAM に記録されるイベントレート、は Detector-FPGA と同様に既に決まっているものであるが、 HK データフォーマットは今回の DE の設計にあたって私が新たに設計したものである。

2.6.1 DRAM に記録されるデータフォーマット

CdTe-DSD にトリガーが発生すると ASIC の数だけ ASIC からフレームが出力される。その際、 Ti や Live Time と言った情報を含む 5 word (20 bytes) のイベントヘッダーが先頭に付与される。その後に 4 枚の ASIC から出力されたフレームが順次に続く。一枚の ASIC から出されるフレーム 2.14 は、DRAM に記録される際に 4 bytes アラインメイトされる。

図 2.15 に DRAM に記録されるデータフォーマットを示す。このデータは header word (0x3C3C0000) と footer word (0x00007777) で挟まれており、以降は DRAM 内で繰り返し記録される。DRAM 内では固定長 (32,768 bytes) を一つの単位としてデータが記録される。以降はこの 32,768 bytes の単位をイベントフレームと呼ぶ。

このイベントフレームは各 CdTe-DSD に付属する Detector-FPGA の DDR2 メモリにリングバッフォーとして記録される。Detector-FPGA はイベントフレームのうち 32000 bytes 以上が書き込まれるたびに、イベントフレームを切り変え、 Detector-FPGA 内部の write pointer を更新する。リングバッファーは 64 MByte の大きさを持っており、Detector-FPGA はデータを受け取るたびに write pointer を更新し、64 MByte に達した場合は先頭に戻ってデータを書き込む。

DE は定期的に write pointer を読み出し、 Detector-FPGA 内部の DRAM からイベントフレームを読み出す必要がある。その度に DE は内部の read pointer を更新し、適切にイベントフレームを読み出す必要がある。Detector-FPGA にイベントフレームを書き込む際、データが読み出されたかの有無の確認は行わず、単純に上書きされる。

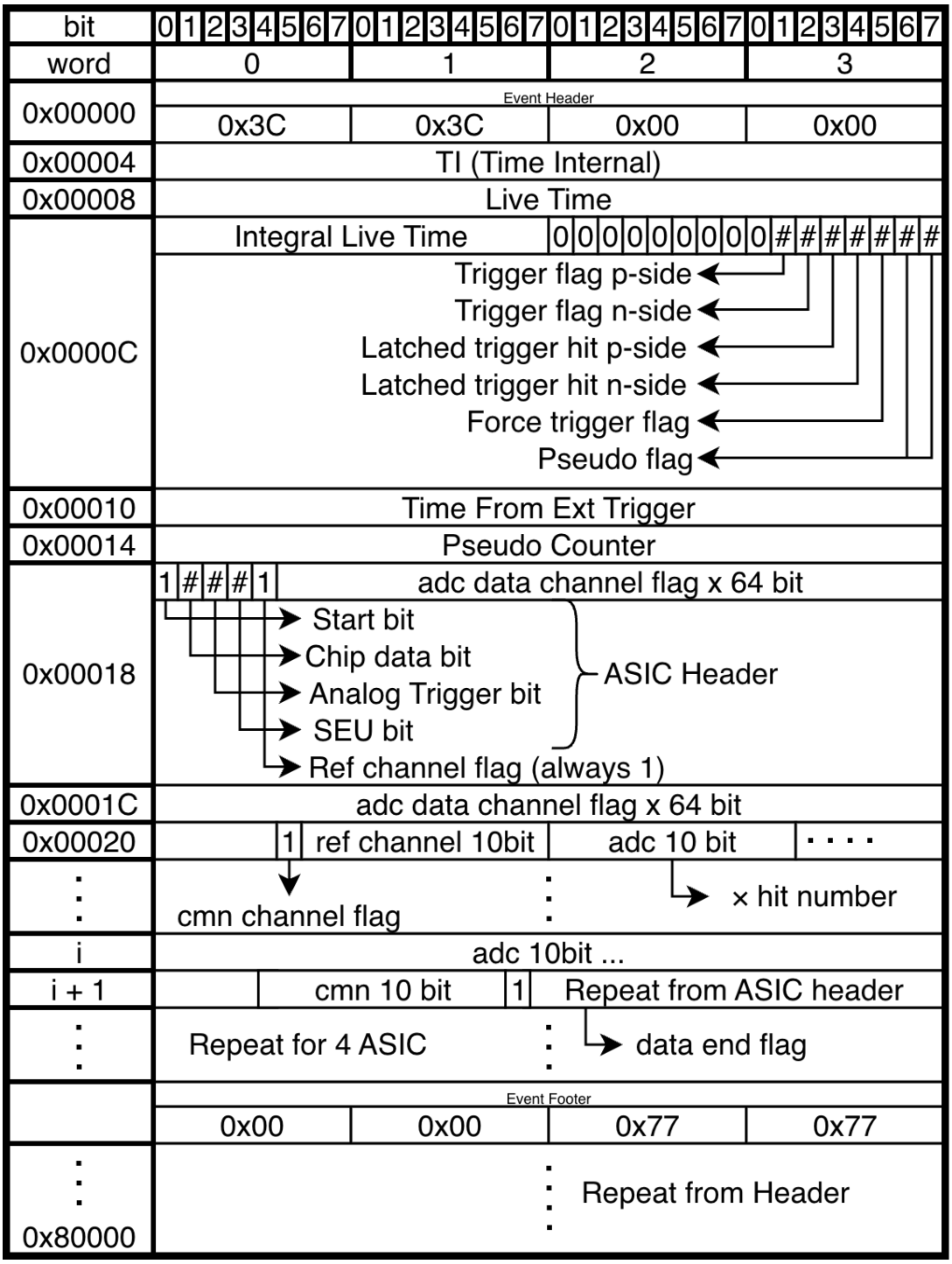


図 2.15: DRAM に記録されるデータのフォーマット (イベントフレーム)

2.6.2 HK データ

HK データは Raspberry Pi を含む DE/DAU 全体の状態を監視するために記録されるデータである。DE 内時刻、DE (Raspberry Pi) の CPU 温度、 CdTe-DSD の温度、CdTe-DSD の湿度、そして各 CdTe-DSD で取得されたイベントの絶対時間を同定するための、タイムコードテーブルなどを付加することができる汎用のデータフォーマットである。

DUA の構造/仕様が確定していないため、具体的な HK データのフォーマットは未定であるが、現段階では Raspberry Pi の CPU 温度と、イベントのタイミングの同定に必要な DE 内での Unix Time と CdTe-DSD 上のタイムコードテーブルのみを保存している。

2.6.3 タイムコードテーブル

CdTe-DSD 上の FPGA が SpaceWire ネットワーク上で出力 (ブロードキャスト) された 6bit/64 Hz のタイムコードを取得した際、 Timecode = 0 のタイミングで FPGA 内部の Ti カウンターをレジスターに保存する。この Timecodea のカウンターと Ti のカウンターは過去 4 個分を保存しており、 Timecode = 0 を受け取る度に更新される。2.16 に CdTe-DSD から取得される Timecode テーブルのフォーマットを示す。この Timecode テーブルは CdTe-DSD からのデータを絶対時間に変換するために必要な情報であり、DE は定期的に CdTe-DSD から Timecode テーブルを最低でも 4 秒に一回読み出し、 HK データとして保存する必要がある。さらに、SpaceWire Router の SpW-FPGA では GPS からの 1PPS 信号を受け取り、そのタイミングでの Ti を記録した 1PPS Table を保存している。2.17 に Timecode + 1PPS テーブルのフォーマットを示す。

bit	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
word	0	1	2	3
0x00	TimeCode Counter 1			
0x04	Local TI 1			
0x08	TimeCode Counter 2			
0x0C	Local TI 2			
0x10	TimeCode Counter 3			
0x14	Local TI 3			
0x18	TimeCode Counter 4			
0x1C	Local TI 4			

図 2.16: TimeCode Table のフォーマット

2.6.4 HK データのフォーマット

HK データは 1024 bytes の固定長フォーマットで保存する。図 2.18 に HK データのフォーマットを示す。HK データは 0xAFAFAFAF のヘッダーで始まり、DE 内時刻 (Unix

bit	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
word	0	1	2	3
0x00	TimeCode Counter 1			
0x04	Local TI 1			
0x08	TimeCode Counter 2			
0x0C	Local TI 2			
0x10	TimeCode Counter 3			
0x14	Local TI 3			
0x18	TimeCode Counter 4			
0x1C	Local TI 4			
0x20	PPS Counter 1			
0x24	Local TI 1 (for PPS signal)			
0x28	PPS Counter 2			
0x2C	Local TI 2 (for PPS signal)			
0x30	PPS Counter 3			
0x34	Local TI 3 (for PPS signal)			
0x38	PPS Counter 4			
0x3C	Local TI 4 (for PPS signal)			

図 2.17: TimeCode + 1PPS Table のフォーマット

Time)、 Raspberry Pi の CPU 温度、そして Router の TimeCode + 1 PPS テーブル x Router 数分、CdTe-DSD の TimeCode テーブル x CdTe-DSD 数分が続く。各 TimeCode テープルは DE のプログラム上で設定した logical address でタグ付けされている。

そのため各 router につき

$$\begin{aligned} &1\ (\text{Header}) + 1\ (\text{LogicalAddress}) + 16\ (\text{TimeCode} + \text{1PPSTable}) \\ &= 18\ \text{words} \end{aligned} \tag{2.2}$$

各 detector につき

$$\begin{aligned} &1\ (\text{Header}) + 1\ (\text{LogicalAddress}) + 8\ (\text{TimeCode} + \text{1PPSTable}) \\ &= 10\ \text{words} \end{aligned} \tag{2.3}$$

を使用する。 1024 bytes (256 words) であるため、 Router 2 台とすると CdTe-DSD は最大 20 台以上の TimeCode テーブルを保存することができる。

bit	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
word	0	1	2	3
0x0000	HK Header			
	0xAF	0xAF	0xAF	0xAF
0x0004	Unix Time in DE			
0x0008	Temperature of DE			
0x000C	Router Timecode Table Header			
	0xBB	0xBB	0xBB	0xBB
0x0010	Logical Address			
0x0014	[TimeCode + 1PPS Table]			
⋮	⋮ Repeat Router Number			
i	Detector Timecode Table Header			
	0xCC	0xCC	0xCC	0xCC
i + 4	Logical Address			
i + 8	[TimeCode + 1PPS Table]			
⋮	⋮ Repeat Detector Number			
j	HK Footer			
	0xDD	0xDD	0xDD	0xDD
⋮	⋮			
0x1000	0			

図 2.18: HK データのフォーマット

第3章　CdTe検出器の読み出しシステム (DE) の開発

この章では今回一から私が開発と実装を行った SuperHERO で使用される CdTe-DSD の読み出しシステムについて説明する。DE は SuperHERO に搭載される CdTe-DSD のデータを収集し、上流の DEU に送信するシステムである。SuperHERO では使用する CdTe-DSD と工学モジュールが既に開発されていたが、それらを制御し、データを収集するシステムは存在していなかった。今回作成した DE はそのシステムにあたり、CdTe-DSD の制御とデータ収集を担う重要なコンポーネントである。

本研究で新規に設計・実装した主な項目は以下の通りである。

- DE/DEU 間通信プロトコルおよびコマンド体系の設計（3.2 節）
- 複数台の CdTe-DSD (DUA) からのデータ収集・集約・転送を行う DE ソフトウェアアーキテクチャの設計と実装（3.3 節）
- 高負荷/異常状態においても安定した動作を実現するための、DE 内部の状態遷移およびスケジューリング手法の設計（3.3 節）
- DE の設計・検証を目的として新規に開発した DUA シミュレータ（第 4 章）

この章では、3.1 節で DE の役割と要求事項について説明する。次に、3.2 節では DE と DEU 間の通信プロトコルについて説明し、次節の 3.3 節では DE のソフトウェア構成について説明し、各要求事項をどのように満たしたかを説明する。最後に、DE の動作の詳細を説明し、SuperHERO でどのように使用されるかを説明する。

3.1　読み出しシステム (DE) の役割と要求事項

SuperHERO には 7 台の CdTe-DSD が搭載される。各 CdTe-DSD には対応する FPGA が存在し、 CdTe-DSD 搭載の ASIC の制御および ASIC から出力されたデータの収集を担う。DE の上流には DEU が存在し、 DE と DEU 間は TCP 通信によって接続されている。DEU は NASA の開発するシステムである。

DE の役割には大きく二つが存在する。一つは上流からのコマンドを各 CdTe-DSD に伝達することである。CdTe-DSD - DE 間では SpaceWire 通信、 DE - DEU では TCP 通信というように通信プロトコルが異なるため、上流からのコマンドを適切に変換し、各 CdTe-DSD に伝達する必要がある。もう一つは各 CdTe-DSD のリングバッファーを監

視し、リングバッファーから戦略的にデータを回収することである。DUA は完全に独立して動作を行いアクティブにデータを送信する機能は存在しない。データはリングバッファーに自動的に貯められていくだけであり、DE は積極的に DUA のリングバッファーにアクセスし、データを回収する必要がある。

DE の設計にあたっては、もう一つの重要な要素として、DEU の開発主体が NASA である点を考慮する必要がある。もし DE を単に SpaceWire 通信を中継するだけのシステムとして設計してしまうと、DEU 側は Detector-FPGA のレジスターや、CdTe-DSD の詳細な仕様を理解した上で、複数台を同時に制御するための複雑なロジックを実装する必要が出てくる。これは、DEU の不要な複雑化を招き、結果として開発コストの増大につながってしまう可能性がある。したがって、DEU 側の処理は可能な限り単純化し、システム全体として役割分担を明確にした上で、各コンポーネントの開発を相互に独立させる設計方針が不可欠である。その抽象化を行った上で DEU を介して DUA が可能とする全ての操作を細かく制御できる柔軟性も必要である。DE の設計にあたって以下の三つの要求事項を定めた。

要求 1: 簡易かつ安全で安定的なプロトコルを規定した上で、DUA の Event データ/HK データの送信および DUA でのエラーを DEU に通知できること

要求 2: 高負荷な状況でも正常に動作し、データの欠損が発生しないこと

要求 3: データ取得中にも緊急のコマンドを受け付け、即座に実行できること

要求 1 は DE と DEU 間の通信プロトコルに関する要求であり、DEU と DE の開発が異なる組織で行われることを考慮したものである。DEU 側で間違ったフォーマットのままコマンドを送信してしまった場合に、誤動作を防止するために、DEU が間違ったフォーマットのコマンドを送信することを何らかの方法で防止する安全性と、DE 側での細かな仕様変更でもインターフェースが壊れないようにする安定性が求められる。その上で通信内容が間違っていたり、エラーが発生した場合に、 DEU 側に適切に通知できることが求められる。

要求 2 は DE の安定性に関する要求である。SuperHERO では長時間にわたり複数台の CdTe-DSD からのデータを収集し続ける必要がある。そのため、 DE は高負荷な状況でも安定して動作し、データの欠損が発生しないことが求められる。例えば、CdTe-DSD の一台のあるチャンネルが発振してしまい、高レートでデータを送信し続けた場合、通常ではありえない高レートのデータが DE に送信されることとなる。その場合、データの読み取り時間をその検出器のみが占有してしまうと、他の正常な検出器のデータを欠損させてしまう可能性がある。このような場合にも、正常な検出器のデータを欠損させることなく収集し続けることが求められる。このような異常事態に対してだけではなく、将来的に数十台にまで CdTe-DSD を増設した場合にもバックグラウンドイベント等の蓄積により、DE が処理するべきデータ量は大きく増加し、そのような高負荷な状況でも安定して動作し続けることが求められる。

要求 3 は DE の応答性に関する要求である。SuperHERO では観測中に緊急のコマンドを DEU から送信し、即座に実行する必要がある場合がある。異常な動作や、検出器の

発振を検出した場合に、即座に検出器を停止させたり、ASIC の設定を変更したりする必要がある。このときに、データ収集の処理によってコマンドの実行が遅延してしまうと、その遅延の分だけバッファーを圧迫し、最悪の場合、データの欠損が発生してしまう可能性がある。そのため、 DE はデータ取得中にも緊急のコマンドを受け付け、即座に実行できることが求められる。

SuperHERO では使用する検出器と、その周辺機器が既に開発されていたが、検出器に指令を送りデータを収集し上流へと送信するシステムは存在していなかった。今回私はその読み出しシステム DE の設計と実装をこの三つの要求事項を満たす形で一から行った。実際にはこれらの要求事項の他にも、細かな設計思想を定め設計を行ったが、その詳細については Appendix B で説明している。

3.2 DE / DEU 間通信インターフェースの規定

通信プロトコルはシステム全体の安定性に大きな影響を与える重要な要素である。TCP 通信と一括りにしてもその上で動作するプロトコルには様々なものが存在し、システムの要件に最適なプロトコルを選択する必要がある。代表例としては HTTP や WebSocket のような汎用的なプロトコルが存在し、さらに Json-RPC や XML-RPC のような RPC フレームワークも存在する。実装コストを許容すれば、 socket 通信に独自の生バイト列から独自のプロトコルを設計することも可能である。しかしこれらのプロトコルの長所短所を考えると、必然的に選択肢を絞ることができる。

HTTP/Websocket は今回の用途には過剰であり、これらの通信をサポートした安定したライブラリはフルスタックのウェブサーバーを内包していることが多く、衛星搭載システムのような限られたリソース上で動作させる場合には過剰である。一方 Json-RPC/XML-RPC はテキストベースであるため、バイナリデータの送受信にオーバーヘッドが大きい。データは生のバイト列で送受信したいが、base 64 エンコードのような変換を行う必要があり、その過程でデータ量が約 1.3 倍に増加してしまう。さらに、独自プロトコルを設計する場合には、通信の仕様を一から設計する必要があり、バグが入り込む余地が大きくなってしまう。特に生のバイト列を数値として解釈する場合には、エンディアンの違いなどにより意図しない動作を引き起こす可能性がある。

そのため、RPC フレームワークとして、データをバイナリ列で送信でき、通信用途に特化してフレームワークが提供されている gRPC プロトコルを採用することにした。この節では 要求 1 を満たすために gRPC プロトコルが適している理由について説明し、私が DE の設計にあたって DE / DEU 間で定義したコマンドについて説明する。

3.2.1 gRPC プロトコルの概要

gRPC は Google が開発した TCP 上で動く RPC (Remote Procedure Call) フレームワークである。実績のある安定した通信プロトコルであり、高負荷なサーバーでの安定して動作することが知られている。gRPC は Protocol Buffers という独自のシリアライズフォーマットを用いて通信を行う。Protocol Buffers では proto ファイルに通信の仕様を定義し、その proto ファイルから型安全な通信コードを自動生成することができる。型安全であるメリットとして、通信の仕様が変更された場合に、コンパイルエラーとして検出できる点が挙げられる。(Appendix C 参照)

DE と DEU 間で共通の proto ファイルを用いることで、通信の仕様を明確に定義することができ、 DE と DEU の開発を独立に行うことができる。さらに proto ファイルは後方互換性が確保できるように設計されており、たとえ使用している proto ファイルが古いものであっても、それが通信の型のサブセットである限り、通信を行うことができる。これは通信時の意図しないクラッシュを防止する上で重要な要素である。

もちろん良いことばかりではなく、gRPC を用いるデメリットも存在する。まず、Protocol Buffers はバイナリフォーマットであり、 JSON や XML よりも高速であるが、純粋なバイナリ通信よりも多少のオーバーヘッドが存在する。もう一つは、サーバーとしての機能として内部にスレッドプールを持っているため、1 対 1 の通信を行う場合を考える

とオーバーヘッドが発生する可能性がある。将来的に組込みシステムやリアルタイム OS 上で動作させることも考えると、gRPC ではなくより軽量な通信プロトコルを用いることも検討する必要がある。しかしながら、 SuperHERO においてはこのオーバーヘッドは無視できる程度であり、gRPC を用いることのメリットが大きいため、 GRPC を採用することとした。

gRPC には以下に挙げる 4 つの異なる通信方法が存在する。

Unary RPC
: 通常の関数呼び出しのように、1 回のリクエストに対して 1 回のレスポンスを返す通信方法。通信は同期的に行なわれ、コマンドの送信と実行完了の通知に最適である。

Server Streaming RPC
: 1 回のリクエストに対して、複数回のレスポンスを返す通信方法。サーバーからの大量のデータ転送に最適である。

Client Streaming RPC
: 複数回のリクエストに対して、 1 回のレスポンスを返す通信方法。クライアントからのデータ送信に最適であり、サーバーは全てのデータを受信した後にレスポンスを返す。

Bidirectional Streaming RPC
: 複数回のリクエストに対して、複数回のレスポンスを返す通信方法。クライアントとサーバーが独立にデータを送受信できる通信方法であり、双方向のデータ転送に最適である。

Unary RPC はコマンドの送信と実行完了の通知に用いられ、Server Streaming RPC は CdTe-DSD からのデータ転送に用いられる。そのため必然的に DE は gRPC サーバーとして動作し、 DEU は GRPC クライアントとして動作することとなる。検出器のセットアップやモード変更などのコマンドは Unary RPC を用い、観測中のデータ転送は Server Streaming RPC を用いることにした。

3.2.2 High Level / Low Level コマンド

DE 及び DAU 全体の状態を変更するコマンドとして、Unary RPC を用いた Low Level コマンドと High Level コマンドの二つのレベルのコマンドを定義した。Low Level コマンドは SpaceWire RMAP コマンドを直接実行するためのインタフェースであり、緊急時などに CdTe-DSD のレジスタを直接操作する目的で用いられる。RMAP コマンドで行なわれる全ての操作を直接実行できるため、Low Level コマンドを用いれば CdTe-DSD のすべての操作を行うことが可能である。一方で、RMAP や Detector-FPGA の詳細な仕様を十分に理解している必要があり、誤った操作を行うと CdTe-DSD が正常に動作しなくなる可能性がある。さらに、後述する TaskQueue と Scheduler の制約により、Low Level コマンドを介して実行される RMAP のコマンドは、実行完了を待つと数十ミリ秒

から一秒程度の遅延が発生する。この Low Level コマンドでは、TaskQueue にコマンドを投入するための RmapReadSubmit および RmapWriteSubmit と、実行結果を取得するための RmapResult の三つのコマンドを定義した。

一方、High Level コマンドは、より抽象化されたインタフェースとして設計しており、複数の FPGA に対する設定をまとめて実行する場合や、DE のモード遷移を行う目的で用いられる。限られたコマンドセットのもとで適切に状態管理を行うことができるため、誤用のリスクを低減し、復帰不可能な状態に陥る可能性を低減することができる。また、High Level コマンドの返り値は適切にエラーを返すように設計しており、コマンドが正常に実行されたかどうかに加え、エラーの原因を通知されるように設計した。以下は定義している High Level コマンドの一覧[1] である。

表 3.1: DE 制御コマンド一覧

コマンド名	説明
`AddDetector` `AddRouter`	CdTe-DSD や SpaceWire Router を DE に追加するコマンド
`RemoveDetector` `RemoveRouter` `RemoveDevice`	CdTe-DSD や SpaceWire Router を DE から削除するコマンド
`GetDetectorList` `GetRouterList` `GetDeviceList`	DE に登録されている CdTe-DSD や SpaceWire Router の一覧を取得するコマンド
`SetVaRegister`	CdTe-DSD の ASIC のレジスタを設定するコマンド
`ConfigureFPMA`	CdTe-DSD の resetwait-time などの FPGA の設定を行うコマンド
`SetLinkSpeed`	SpaceWire のリンク速度を設定するコマンド
`GetDEStatus`	DE がどの状態にあるかを取得するコマンド
`StartDataStream`	Observation モードに遷移し、CdTe-DSD からデータ収集を行い DEU へデータ送信を開始するコマンド
`StopDataStream`	Observation モードを終了し、Idle モードに遷移するコマンド

表 3.1 に示されているコマンドの実際の使用例については、3.4 節で説明するが、簡単な流れとしては、まず `AddDetector`、`AddRouter` コマンドで使用する CdTe-DSD や SpaceWire Router のロジカルアドレスと、そのパスアドレスを DE に登録する。その次に、`SetVaRegister` や `ConfigureFPMA` コマンドで各 CdTe-DSD の ASIC や FPGA の設定を行い、最後に `StartDataStream` コマンドで Observation モードに遷移し、データ収集を開始、データ収集を終了する場合には `StopDataStream` コマンドを送信する。

[1] 設計初期段階では、 High Level コマンドは TaskQueue に積む `HLSubmit` コマンドと、TaskQueue からの実行結果を取得する `HLResult` コマンドの二種類のみを定義していた。しかしこの仕様だと実装の複雑化を招くため、各コマンドごとに別々の RPC を定義する形に変更した。詳細は Appendix I.1 にて説明している。

3.2.3 データ転送

DE から DEU へのデータ転送には Server Streaming RPC を用いる。もしデータ転送に Unary RPC を用いてしまうと、観測ルーチン中では DE は自動でデータを収集し続けるだけとなり、DEU が明示的にデータを要求する必要がでてきてしまう。そのため、実観測では、どんなデータレートでデータが収集されるか予測し、その上で適切なタイミングでデータを要求するという挙動が DEU に求められてしまう。要求 1 の簡潔なプロトコルを規定するという要求からも、DE から DEU へ自動的にデータを送信し続け、データが欲しいときに DEU が受信することができる Server Streaming RPC を用いることにした。

データ転送は `DataStream` という Server Streaming RPC で実装しており、DEU はこの RPC を呼び出すことで、 DE からのデータを受信し続けることができる。そのストリーミング中では、次のデータが到着するまで DEU はブロックされるため、DEU は複雑なポーリングを行う必要がなく、シンプルにデータを受信し続けることができる。観測スケジュールを DE 側に押し付けることができるため、DEU だけではなく DE を操作し CdTe-DSD の操作を行う単一のソフトウェアの開発 AppendixF の開発も容易になった。

3.3 DE のソフトウェアの構成

今回私は DE のソフトウェアの構成を含め、一から DE のソフトウェアを設計し、実装を行った。DE のソフトウェアは C++ で実装しており、 Raspberry Pi OS 上で動作する。Raspberry Pi OS は NixOS/Linux ベースの OS であり (Appendix E 参照)、 DE のソフトウェアは systemd サービスとして自動起動するように設定している。この節では C++ のクラス図を用いて、私の設計した DE のソフトウェアの構成について説明を行う。

3.3.1 DE のクラス図と主要コンポーネント

図 3.1 に簡略化した DE のクラス図を示す。DE のソフトウェアを構成する主な要素は `Scheduler`, `TaskQueue`, `DataRecorder`, `HKRecorder` の四つである。`Scheduler` は CdTe-DSD からのデータ収集をスケジューリングするクラスであり、データ収集のオペレーションでのみ動作する。`TaskQueue` は DE に送られてきたコマンドを順次実行する機能があり、gRPC サーバーから送られてきたコマンドは全て `TaskQueue` に投入され、別のスレッドで順次実行される。この実装により gRPC サーバーのスレッドとメインスレッドを分離し、 DE の安定性を向上させている。`DataRecorder` は CdTe-DSD からの観測データを収集し、 DEU に送信するクラスであり、`HKRecorder` は CdTe-DSD からの HK データを収集し、 DEU に送信するクラスである。以降の説明では、これらの主要コンポーネントを中心に説明を行う。

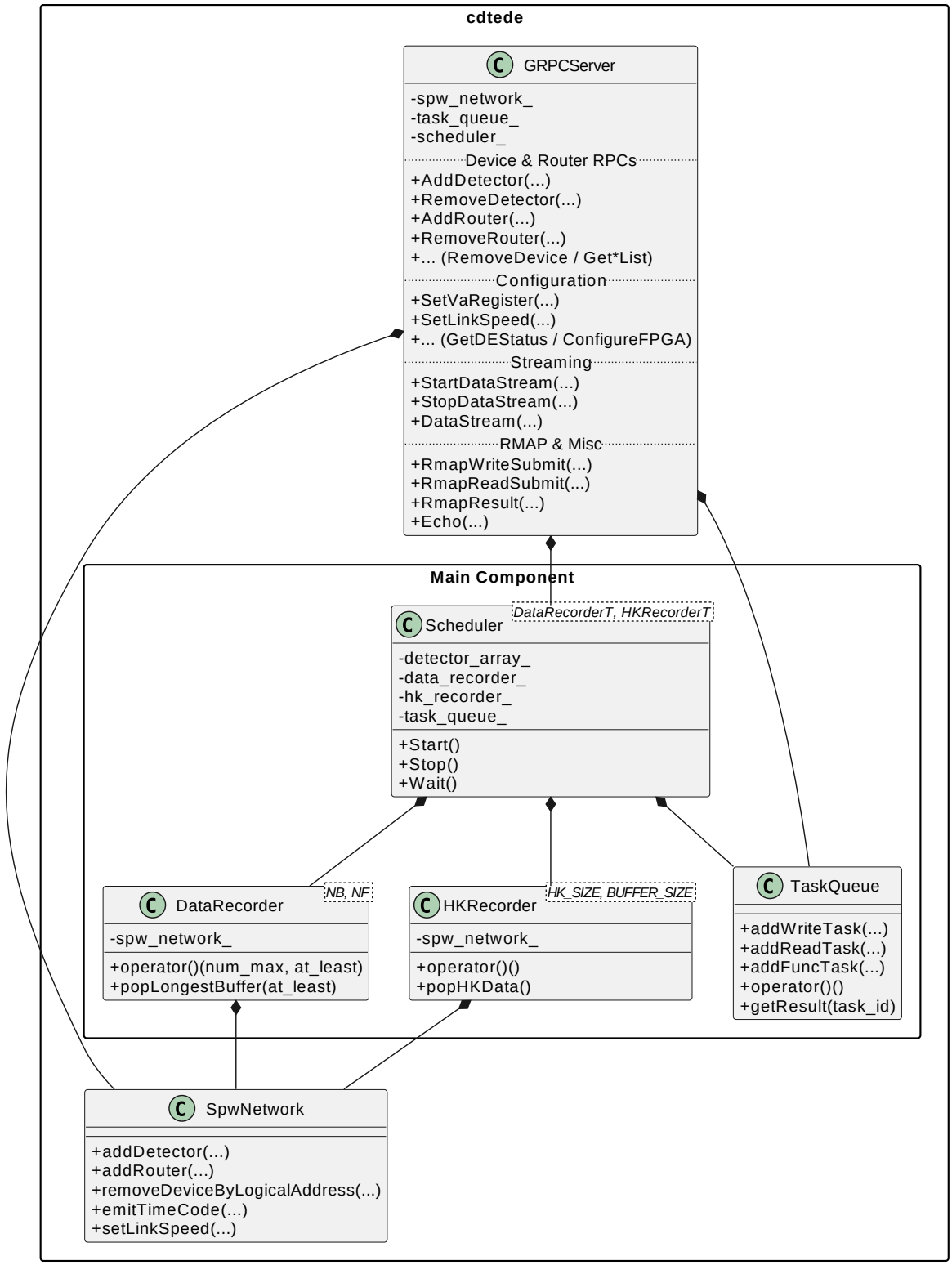

cdtede
GRPCServer
-spw_network_
-task_queue_
-scheduler_
Device & Router RPCs
+AddDetector(...)
+RemoveDetector(...)
+AddRouter(...)
+RemoveRouter(...)
+... (RemoveDevice / Get*List)
Configuration
+SetVaRegister(...)
+SetLinkSpeed(...)
+... (GetDEStatus / ConfigureFPGA)
Streaming
+StartDataStream(...)
+StopDataStream(...)
+DataStream(...)
RMAP & Misc
+RmapWriteSubmit(...)
+RmapReadSubmit(...)
+RmapResult(...)
+Echo(...)
Main Component
Scheduler
DataRecorderT, HKRecorderT
-detector_array_
-data_recorder_
-hk_recorder_
-task_queue_
+Start()
+Stop()
+Wait()
DataRecorder
NB, NF
-spw_network_
+operator()(num_max, at_least)
+popLongestBuffer(at_least)
HKRecorder
HK_SIZE, BUFFER_SIZE
-spw_network_
+operator()()
+popHKData()
TaskQueue
+addWriteTask(...)
+addReadTask(...)
+addFuncTask(...)
+operator()()
+getResult(task_id)
SpwNetwork
+addDetector(...)
+addRouter(...)
+removeDeviceByLogicalAddress(...)
+emitTimeCode(...)
+setLinkSpeed(...)

図 3.1: DE のクラス図

3.3.2 DE のモード遷移

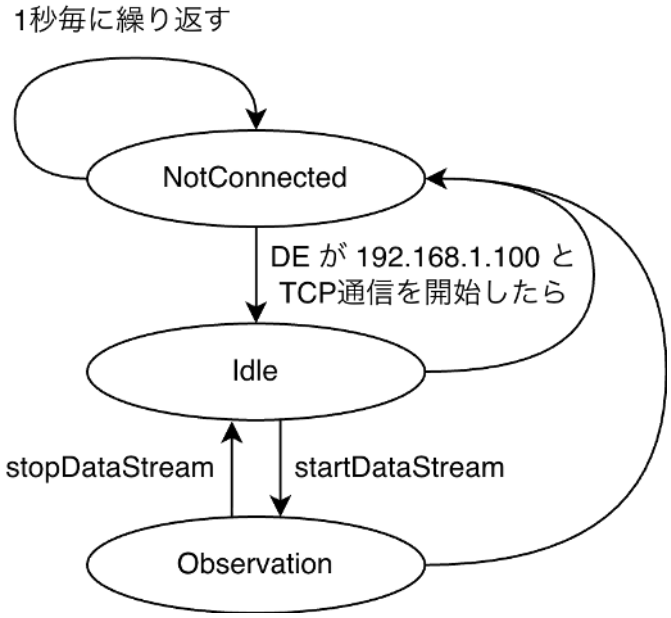


図 3.2: DE のモード遷移図

図 3.2 に示しているように、DE は NotConnected、 Idle、 Observation の三つのモードを持っている。NotConnected モードは SpaceWire Router に未接続な状態であり、gRPC サーバーは起動しているが、CdTe-DSD とやり取りをするコマンドは受け付けない。1 秒毎に SpaceWire Router への接続を試み、接続に成功した場合に Idle モードに遷移する。この状態は図 3.1 で、`GRPCServer` が SpwNetwork のみにアクセスしている状態に対応する。SpwNetwork の初期化のみを試み、他の操作は行わない。NotConnected モードは、もっとも基本的なモードであり、Idle モードと Observation モード時に異常が発生した場合にも遷移し、接続を再確立しようと試みる。

次の Idle モードでは SpaceWire Router との接続が確立されており、 High Level/Low Level コマンドを受け付ける。gRPC 経由で来たコマンドは `TaskQueue` に積みまれ順次実行される。図 3.1 のうち `TaskQueue` のみにアクセスしている状態に対応し、gRPC からのリクエストを `TaskQueue` に投げ、別のスレッドで `TaskQueue` を消費している状態である。

最後の Observation モードでは CdTe-DSD からのデータ収集を行い、 DEU にデータを送信する。図 3.1 のうち `Scheduler` が `Start()` している状態に対応する。`Scheduler` は TimeCode と データ収集の 2 つの thread を起動し、TimeCode を検出器に送信しながら、データ収集を行っている。

DE のプログラムは Raspberry Pi が起動したときに自動的に起動する。そのため、SpaceWire Router と常時接続されており、その接続が切れない限り Idle モードと Observation モードを行き来することになる。Idle モード及び、NotConnected モードでは gRPC サーバーからのアクセスや DE 本体の負荷は非常に軽いものとなっている。その

ため、ここでは複雑な戦略を用いず、 `TaskQueue` ベースでコマンドを順次実行する方式を採用した。

3.3.3 要求事項を満たすデータ収集方法の確立

要求 2、要求 3 は主にデータ取得に関するものである。そして、これらの要求事項を満たすために、`Scheduler`、 `DataRecorder`、 `HKRecorder` が以下のような観測要求事項を満たすように設計した。

観測要求 1 (周期性) Write Pointer を定期的に読み出し、 CdTe-DSD からデータを取得すること

観測要求 2 (公平性) 複数台の CdTe-DSD から均等にデータを取得すること

観測要求 3 (同期性) TimeCode に同期し HK データを 1 秒間隔で定期的に取得すること

観測要求 4 (緊急性) データ取得中に上流からのコマンドを受け付け実行すること

観測要求 5 (柔軟性) Write Pointer の読み出し頻度を動的に調整し、 DE の負荷を抑えつつデータ欠損を防止すること

観測要求 1 は CdTe-DSD からのデータ取得の基本的な要件である。Detector-FPGA のデータ取得中は、DUA からデータ取得完了の通知などは一切行われない。そのため、DE は定期的に DUA の Detector-FPGA にアクセスし、write pointer を読み出すことでデータがどこまで書き込まれたかを確認する必要がある。

観測要求 2 は複数台の CdTe-DSD から公平にデータを取得するための要件である。DE は最大 7 台の DUA からのデータを同時に受け取ることとなる。そのため、`DataRecorder` では、いかに公平に各 DUA からデータを取得するかが重要な要素となる。もしトリガーがバックグラウンドによって発振するような検出器の不具合が起った場合に、一つの検出器のデータレートが高くなってしまう可能性がある。その場合に、一つの検出器ばかりからデータを取得してしまうと、他の検出器の DRAM がオーバーフローし、データの欠損が発生する可能性があるからである。

観測要求 3、観測要求 4 はデータ取得以外のデータ取得中に DE が満たすべき要件である。DE は観測データ取得だけではなく、データ取得中に送信されている TimeCode に同期されて記録された TimeCode テーブルとそれを含む HK データ (2.6.2 節参照) を取得する必要もある上にデータ取得中に上流からのコマンドを受け付け、実行する必要もある。TimeCode テーブルは TimeCode がゼロのタイミングで更新されており、このテーブルには 4 つの TimeCode に対応する観測機内時間 TI が格納されている。そのため、最低でも 4 秒に一回は取得する必要があり、理想的には 1 秒に一回取得することが望ましい。

観測要求 5 は DE の負荷を抑えつつデータ欠損を防止するための要件である。DE は地上試験での 1 台高レート構成や、実観測を見越した 7 台の高レート観測、そしてデータの取得方法によって、一秒に数イベントから数万イベントまで幅広いデータレートに対応

する必要がある。そのため、DE はデータ取得中に write pointer の読み出し頻度を動的に調整し、DE の負荷を抑えつつデータ欠損を防止する必要がある。動的な調整方法を採用することで、 DE のソフトウェアや設定パラメータを変更することなく、幅広いデータレートと観測シナリオに対応することが可能となる。

3.3.4 DataRecorder

DataRecorder は 観測要求 1 及び、観測要求 2 の要件を達成するための重要なクラスである。その根幹は、 Detector-FPGA の write-pointer を周期的に同期し、データを公平に取得することにあり、そのために以下に示すアルゴリズム 1 を設計した。

アルゴリズム 1 DataRecorder のデータ取得アルゴリズム

$FRAME\_SIZE \leftarrow 32768$
$CdTeDRAM\_SIZE \leftarrow 67108864$
$read\_buf\_index$ は前回の位置を保持
$detectors$ は CdTe-DSD 抽象クラスの配列

Input:
num_max ▷ 1 回の呼び出しで読み出す最大フレーム数
at_least ▷ 選択対象とするために必要な最小フレーム数

function DataRecorder(num_max, at_least)
    $selected\_detector \leftarrow$ null
    $num\_frames \leftarrow 0$
        ▷ Round-robin: 同期 → 判定 → 選択
    for $k \leftarrow 1$ **to** $NUMBER\_OF\_BUFFERS$ **do**
        $read\_buf\_index \leftarrow (read\_buf\_index + 1) \bmod NUMBER\_OF\_BUFFERS$
        $detector \leftarrow detectors[read\_buf\_index]$
        sync_write_pointer($detector$) ▷ write_ptr を最新化
        if $detector.getDistance() < at\_least \times FRAME\_SIZE$ **then**
            continue ▷ 十分に溜まっていなければ次へ
        end if
        $selected\_detector \leftarrow detector$
        $num\_frames \leftarrow \min\left(\left\lfloor \frac{detector.getDistance()}{FRAME\_SIZE} \right\rfloor, num\_max\right)$
        break ▷ 今回読む検出器を確定
    end for
        ▷ 選択された検出器から連続読み出し
    Read($selected\_detector$, num_frames)
end function

DataRecorder の `operator()(num_max, at_least)` 関数は、引数として一回の呼び出

しで読み出す最大フレーム数 num_max と、選択対象とするために必要な最小フレーム数 at_least を受け取る。一回の呼び出しでこの関数は at_least フレーム以上、最大 num_max フレームまでのフレームを連続的に読み出す。

アルゴリズムとしては、巡回しながらデータ収集をする round-robin 方式を採用している。まず始めに、前回読み出した CdTe-DSD の次の CdTe-DSD から順に write pointer を同期する。(観測要求 1) write pointer の同期は Detector-FPGA に RMAP コマンドを送り、最新の write pointer を取得することで行う。write pointer の同期が終わると、DataRecorder 内部の read pointer との差分を計算し、その差分が at_least フレーム以上である CdTe-DSD を発見すると、その CdTe-DSD を読み出し対象として選択する。選択された CdTe-DSD からは、 write pointer と read pointer の差分に基づき、最大 num_max フレームまで連続して読み出しを行う。

巡回しながらデータ収集を行うことで、各 CdTe-DSD は num_max フレームずつ順番に読み出されることが保証される。(観測要求 2) また、読み出し対象の CdTe-DSD 選択時に write pointer を同期するため、write pointer が同期されないまま読み出しルーチンに入ることはない。特に、 round robit 方式の利点としては各 CdTe-DSD からのデータ取得レートが同一と予測される場合に、効果を発揮する。このような場合、パラメータと待機時間を適宜調整することで、write pointer 同期時に予測される読み出しフレーム数を高い確度で at_least フレーム以上、 num_max フレーム以下に抑えることができる。これが意味するところは、 write pointer の同期が無駄なく行われることであり、DE の負荷を抑えつつ、データ欠損を防止することが可能となる。

3.3.5 HKRecorder

HKRecorder は 観測要求 3 の HK データ収集を担当するクラスで、2.6.4 節で述べた HK データを CdTe-DSD から取得する。HK 収集はデータ収集と並行して行われるものであるが、観測データと比べてデータレートが非常に低い。さらに TimeCode テーブルは 1 秒ごとに全 DUA と Router で更新されることが保証されているため、HKRecorder のアルゴリズムは呼び出し時に全 DUA と Router から HK データを取得するという単純なものである。

3.3.6 Scheduler

Scheduler は CdTe-DSD からの観測データ収集、および HK データ収集をスケジューリングするクラスである。それと同時に観測要求 3、観測要求 4、観測要求 5 を満たすための重要なクラスである。

Scheduler 内でのタスクのスケジューリングと戦略が DE の性能に大きく影響を与える。DataRecorder / HKRecorder / TaskQueue の呼び出しの順序や頻度によって、DE の CPU 使用率や CdTe-DSD からのデータ欠損の発生率が大きく変化するからである。そのため、Scheduler には高負荷な状況でも DUA に緊急コマンドを送信できるようにすること、そしてデータの欠損を最小限に抑えることが求められる。

Scheduler には二つのスレッドが存在し、一つは SpaceWire ネットワーク上に TimeCode を送信する TimeCode Thread、もう一つは CdTe-DSD からのデータ収集を行う Data Collection Thread である。TimeCode Thread は 64 Hz の周期で SpaceWire ネットワーク上に TimeCode をブロードキャストする。 設計初期では、この TimeCode に同期してデータ収集を行う方式を検討していた[2] が、SuperHERO では TimeCode に従うのではなく、独立にデータ収集を行う方式を採用している。これには以下に挙げる二つの理由がある。

まず一つ目の理由として、CdTe-DSD のデータ生成方式がイベント駆動型であることが挙げられる。CdTe-DSD のようなトリガ駆動型の観測器では、TimeCode に同期してデータ収集を行っているわけではなく、したがって TimeCode に基づいてデータ収集を行う必然性は小さい。さらに、トリガ駆動であるという特性から、CdTe-DSD で生成されるデータ量を事前に予測することは困難である。特にコンパクト天体の観測では、観測レートが桁で変化することもあり得るため、TimeCode に基づいてデータ収集を行う方式では、幅広いデータレートに対応することが難しい。

二つ目の理由として、DE と Router 間が TCP により接続されている点が挙げられる。TCP は信頼性の高い通信プロトコルである一方、パケットの到着時間に揺らぎ（ジッター）が生じることが知られている。5 節で実際に測定した結果によれば、短いパケットのやりとりでは、平均の通信時間の二倍程度のジッターが発生することが確認されている。そのため、TimeCode に同期してデータ収集を行う場合、CdTe-DSD におけるデータ収集処理時間に通信ジッターを含む遅延の総和が、TimeCode の周期（64 Hz、すなわち約 15.6 ms）に対して十分に小さく抑えられている必要がある。この条件を満たすためには、TimeCode の間隔に数十パーセント程度の安全マージンを設ける必要があり、その結果として最大のデータ収集レートが制限されてしまう。

SuperHERO では TimeCode に依存しない非同期型のデータ収集方式を採用し、ネットワーク条件や観測状況の変動に対して柔軟かつ安定した運用を可能としている。設計初期では TimeCode のスレッドとデータ収集のスレッドが同時に動作する方式を採用していた。そのため、TimeCode の送受信に大きく遅延をさせないために、データ収集時の通信では最大の遅延を考慮して、毎回次の TimeCode までの時間と比較しつつデータを収集し、TimeCode の送受信に待ち合わせるような設計を行っていた。また、write_pointer の同期は遅延の観点で予測が難しく、必然的に固定のポーリング間隔で行う必要があった。そのため、セットアップや台数を変更するたびに、適切なポーリング間隔を設定する必要があり、ソフトウェアを DE にデプロイし直す必要があった。

アルゴリズム 2 に示すのは Scheduler のメインループの擬似コードである。ループ内では常に最初に TaskQueue のタスクを全て消費してからデータ取得を行う、TaskQueue のタスクは DE の観測中にはほとんど発生しないので、通常はループ中では遅延なく消費されることが期待される。一方で TaskQueue にタスクが溜まる場合は、 Scheduler を停止するコマンドや緊急時に送信された Low Level コマンドが含まれる可能性があるため、優先的に処理される必要があるため、ループの最初で確実に消費されるようにしている。(観測要求 4)

[2] Appendix I.2 に初期の設計で実際にどのような問題が発生したかを説明している。

アルゴリズム 2 Scheduler のメインループ

$wait\_time\_min \leftarrow 100$ ▷ 最小待機時間 [μs]
$wait\_time\_max \leftarrow 1{,}000{,}000$ ▷ 最大待機時間 [μs]
$wait\_time \leftarrow wait\_time\_min$
$N \leftarrow$ detectors.size() ▷ 検出器台数
$M \leftarrow 4$ ▷ 各検出器から 1 周で最大 M 枚読み出すハイパーパラメータ（$M > 1$）
while running **do**
    TASK_QUEUE.CONSUMEALL ▷ キューに溜まったタスクをすべて消費
    if $hk\_recorded = false$ **then**
        HKRECORDER
        $hk\_recorded \leftarrow true$ ▷ 当該サイクルでの HK 重複取得を防止
    end if
    $high\_load \leftarrow false$ ▷ 高負荷検出フラグ
    $frames\_drained \leftarrow 0$ ▷ 当該ループで回収した総フレーム数
    for $i \leftarrow 1$ **to** N **do**
        $f \leftarrow$ DATARECORDER(M) ▷ 検出器 i から最大 M 枚読み出し
        ▷ f は実際に読み出したフレーム数
        if $f = M$ **then**
            $high\_load \leftarrow true$ ▷ 単一検出器で上限まで取得 ⇒ 局所的高負荷
        end if
        $frames\_drained \leftarrow frames\_drained + f$
    end for ▷ 1 周で最大 $M \times N$ フレームを回収
    if $frames\_drained > 2N \vee high\_load$ **then** ▷ 全体量/局所量で高負荷検出
        $wait\_time \leftarrow \max\left(\left\lfloor \frac{wait\_time}{2} \right\rfloor,\ wait\_time\_min\right)$
        YIELD ▷ 待機時間を短縮し即座に再スケジュール
    else if $frames\_drained > N$ **then** ▷ 中負荷
        SLEEP_FOR(wait_time μs)
    else ▷ 低負荷
        $wait\_time \leftarrow \min(2 \times wait\_time,\ wait\_time\_max)$
        SLEEP_FOR(wait_time μs)
    end if
end while

HK データの取得は、別スレッドから制御されるフラグ `hk_recorded` によって管理される。`hk_recorded` は、確実に CdTe-DSD が TimeCode = 0 を受信し TimeCode テーブルが更新された状態であることを保証するため、TimeCode = 1 のタイミングで false に設定される。これにより、各周期における HK 取得の実行可否が厳密に制御される。(観測要求 3)

Scheduler は、各ループにおいて DataRecorder を用いたラウンドロビン方式の読み出しを行い、同時に負荷状況に応じて待ち時間 `wait_time` を自動調整する。調整方式は実装および解析の容易性を優先し、単純なフィードバック則に基づいて設計している。ハイパーパラメータを `N`（通常は CdTe-DSD の台数）とし、各ループで実際に読み出すことができたフレーム総数を `frames_drained` と定義する。各ループでは最大 `N` 回の DataRecorder 呼び出しを行い、各呼び出しにおいて最大 2 フレームずつ読み出す。したがって、1 周期あたりに理論上読み出し可能なフレーム総数は最大で `2N` となる。`frames_drained` は、この上限に対する実効的な読み出し量を表し、Scheduler の負荷推定指標として用いられる。

負荷判定と待ち時間調整の規則は次の三段階で定義される。

(1) $frames\_drained > 2N$ の場合：全ての DataRecorder 呼び出しにおいて最大数のフレームが読み出されたことを意味する。すなわち、全 CdTe-DSD においてデータ生成が活発であり、DE の負荷が高い状態にあると判断される。この場合、wait_time を 1/2 に減少させ、下限 wait_time_min を下回らないように制限する。また、遅延を極小化するため sleep は行わない。

(2) $frames\_drained > N$ の場合：各 CdTe-DSD からある程度のデータが読み出されている状態であり、負荷は中程度であると判断される。この場合、現在の wait_time が適切であるとみなし、値は変更せず、wait_time だけ sleep を行う。

(3) $frames\_drained \leq N$ の場合：ほとんどの CdTe-DSD から十分なフレームが得られていない状態であり、DE の負荷は低いと判断される。この場合、wait_time を 2 倍に増加させ、上限 wait_time_max を超えないように制限したうえで sleep を行う。

このアルゴリズムにより、Scheduler は取得可能なフレーム数に応じて待ち時間を動的に調整し、高負荷時には積極的に読み出しを行い、低負荷時には不要なポーリングを抑制する挙動をとる。その結果として、DE 全体としての計算資源の利用効率と、データ取得の遅延の双方を一定程度最適化することが可能となる。(観測要求 5)

3.4 DE を使用した観測の流れ

DE を用いた観測の全体的な制御フローを図 3.3 に示す。本システムでは、観測に必要な操作の大部分が High Level コマンドのみで完結するよう設計している。

DE は電源投入と同時に起動し、初期状態として NotConnected モードに遷移する。NotConnected モードでは SpaceWire Router への接続処理のみを実行し、接続が確立した時点で Idle モードに遷移する。なお、いずれのモードにおいても gRPC サーバーは常時起動しており、クライアントからの接続要求を受け付けることが可能である。ただし、NotConnected モード中は High Level コマンドおよび Low Level コマンドの受け付けは行わず、Router との接続確立が完了するまで外部からの制御は無効化される。

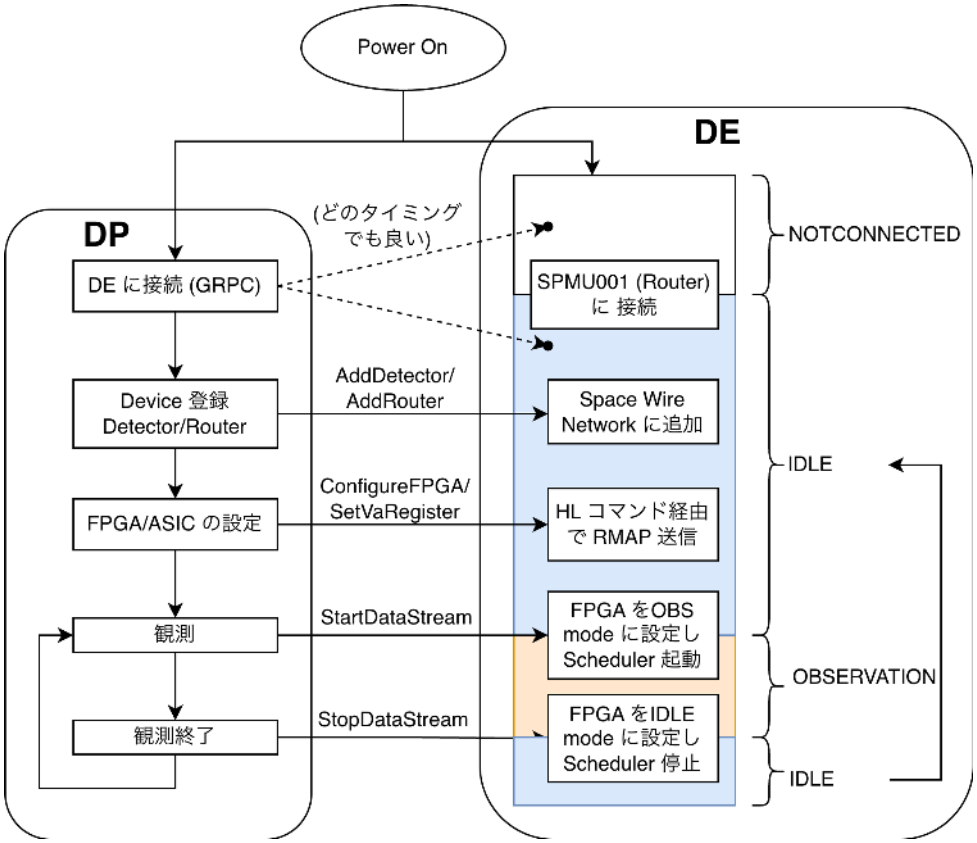


図 3.3: DE を用いた観測の流れ

Idle モードに遷移した DE は、gRPC 経由で送信される High Level/Low Level コマンドを受理可能な状態となる。Idle モード中に High Level コマンドを用いて SpaceWire Network に SpaceWire Router デバイスの登録、および CdTe-DSD デバイスの登録を行う。また同時に、CdTe-DSD の ASIC および FPGA に対する設定を実施する。加えて、SpaceWire Router のリンクスピードの設定などの通信系パラメータの調整もこの段階で行う。将来的には、CdTe-DSD に印加する高電圧バイアスの設定についても、同様に High Level コマンドから制御可能とする予定である。

観測を開始する際には、クライアントから High Level コマンド `StartDataStream` を送信する。これを受信した DE は Scheduler を起動し、Observation モードに遷移する。Observation モードでは、CdTe-DSD からの観測データの収集および HK データの収集が同時に開始される。また、同期してクライアント側では gRPC ストリーミングを用いたデータ受信が開始され、観測データおよび HK データがリアルタイムに送信される。

Observation モードにおいて High Level コマンド `StopDataStream` を受信した場合、DE は Scheduler を停止し、Idle モードへ遷移する。これに伴い、gRPC ストリーミングに対して終了通知が送信され、クライアント側には観測データおよび HK データの送信終了が通達される。

3.5 まとめ

本章では CdTe-DSD 用 DE (Detector Electronics) のソフトウェア設計について述べた。SuperHERO ミッションでは使用する DUA とミラーモジュールが決定していたが、それらを繋ぐ DE のソフトウェアは新規に開発する必要があり、今回 CdTe-DSD 用 DE のソフトウェア設計を一から行い、完成させた。

DE のソフトウェアには、要求 1 から 要求 3 を設け、それらを満たすための設計方針と実装について詳細に説明した。要求 1 に対応するために、DE のソフトウェアは High Level コマンドと Low Level コマンドの二層構造を採用した。 High Level コマンドは抽象化されたインタフェースを提供し、誤用のリスクを低減する一方で、 Low Level コマンドは柔軟な制御を可能とする。この二層構造により、DE の安定性と柔軟性の両立を実現した。

要求 2 および 要求 3 に対応するために、DataRecorder、 HKRecorder、 Scheduler の三つの主要なクラスを設計した。DataRecorder はラウンドロビン方式による公平なデータ取得を実現し、HKRecorder は TimeCode に同期した HK データの定期取得を行う。Scheduler はこれらのデータ取得を統合し、負荷に応じた待ち時間調整を行うことで、高負荷時にも緊急コマンドの処理を可能とし、データ欠損を最小限に抑えることを達成した。

第4章　DUA シミュレータの作成

DE の開発に付随し DUA シミュレータを作成した。本章では DUA シミュレータの開発について述べる。4.1 節では DUA シミュレータの必要性について述べ、4.2 節ではその必要性を満たす DUA シミュレータの構成について述べる。4.3 節では DUA シミュレータが生成する模擬データの生成方法について述べ、最後に 4.4 節では DUA シミュレータの実際の運用について述べる。

4.1　DUA シミュレーターの必要性

DUA ミュレータは NASA が DEU の開発を進めるにあたって必須のものである。NASA の開発者が DEU を設計し動作テストを行うにあたって、CdTe-DSD を 7 台用意して実際に動作させることはコスト的に非常に困難である。特にデータの同時取得を模擬する場合には、7 台全てを冷却した上で放射線ビームを照射しなければならず、大掛かりな実験設備を用意できたとしても、実験の準備に多大な時間と労力がかかってしまう。そのため、DUA の動作を模擬するシミュレーターを作成し、NASA 側で DEU の設計・動作テストを行なえるようにする必要がある。

DEU の役割は CdTe-DSD からのデータを取得し、地上に送信することである。そして、地上に送られたデータはリアルタイムで解析され Quick Look 解析が行われる。従って、DUA シミュレーターでは Quick Look プログラムを作成するために十分にリアルなデータを生成できる必要がある。さらに DEU 運用のテストでは、様々な観測シナリオに応じたテストも必要となる。そのため、イベントのレートを変化させたり、特定のエネルギー領域に集中したイベントを生成したりするなど、多様な観測シナリオに応じたデータを生成できる柔軟性も求められる。

この二つに追加して、 DUA シミュレーターは DE のコードを一切変更せずに動作させることができる必要がある。これは、必須要項ではないが DUA シミュレータを変更する際に DE のコードを変更してしまうと、動作テスト段階で確認したはずの部分にバグが混入してしまうリスクがあるためである。そのため、 実運用のテスト段階でも使用可能な DUA シミュレーターを作成するためには、DE のコードを一切変更せずに動作させることができる必要がある。

以上をまとめると、DUA シミュレーターには以下の 3 つの要求が存在する。

シミュレーター要求 1: リアルなデータと同等のデータを生成できること。

シミュレーター要求 2: 観測シナリオに応じたデータを生成する柔軟性を持つこと。

シミュレーター要求 3: DE のコードを一切変更せずに動作させることができること。

4.2 DUA シミュレーターの構成

DUA シミュレーターとして取れる戦略は大きく分けて二つ考えられる。一つは DE 自体にシミュレータの機能を組み込む方法であり、もう一つは DUA と同様のインターフェースを持つ独立したシミュレータを作成する方法である。

前者の解決方法に則る場合、DE 内部の SpaceWire のインターフェースを Mock する方法が考えられる。これは単純な構成で単体のソフトウェアとして動作させることができるが、DE 起動時にシミュレータかどうかを選択するためのコマンドやインターフェースを追加する必要がある。なぜならば、シミュレータの挙動を細かく制御するにしても、DE が提供するインターフェースが gRPC のみであるため、DUA シミュレータの挙動を変更するにも DEU からの通信に頼る必要があるためである。この時点で シミュレーター要求 3 を満たすことができなくなっているが、さらに問題がある。例えば、気球実験本番直前など DE のソフトウェアが変更できなくなった場合、DE からシミュレーターの機能を取り除くのは新たなバグを混入させるリスクがあるため困難である。そのため、気球実験の際もシミュレータの機能を組み込んだまま DE を使用することとなる。この場合、間違ってシミュレータの機能が動作したまま気球実験を行なってしまうリスクがあり、正しく観測が行なえなくなる可能性がある。直ちに気付けるようなデータであれば良いが、 シミュレーター要求 1 を満たすようなリアルなデータを生成する場合、シミュレータの機能が動作していることに気付けない可能性もあり、限られた観測時間を無駄にしてしまうリスクがある。

そのため、後者を採用し、プログラム単位で DE のソフトウェアを隔離した上で、異なるデバイスとして DUA シミュレータを構成することとした。DE のプログラムは TCP 通信をするだけであるため、全く同じプログラムを使用しつつ、 DUA シミュレータと接続することができる。これは シミュレーター要求 3 を満たす上で非常に有効である。さらに、このようにするこで本番には DUA シミュレータとして使用していた外部デバイスを取り外すだけで、DUA シミュレータを含まない純粋な DE の状態に戻すことができるため、気球実験本番で誤ってシミュレータの機能が動作するリスクを排除することができる。

図 4.1 に DE シミュレータの構成を示す。DUA シミュレータは SpaceWire Router と同じ IP アドレスとポート番号を持ち、DE からの TCP 通信を受け取る。その上で SpaceWire Router と同じ SSDTP2 の TCP 通信プロトコルに従って DE からのコマンドを解釈し、応答を返す。TCP 通信を模擬することで、DE の状態を変えるためのコマンドやインターフェースを追加する必要がなく、DE のソフトウェアも一切の変更を加えずに動作させることができる。さらに、シミュレータの挙動はその別のソフトウェアの調整を通じて細かく制御することが可能であり、実運用では、シミュレータを別のデバイス上で動作をさせる。

別デバイス上で動作させることは、 シミュレーター要求 2 を満たす上で重要である。別デバイスでかつ、独立したソフトウェアとして動作させることで、コマンドラインや設定ファイルを通じてシミュレータの挙動を細かく制御することができる。現在はイベントレートをコマンドライン引数で指定できる程度の柔軟性しか持たないが、将来的には特定のスペクトルを模擬したり、特定のパターンでイベントを生成したりするなど、多様な観

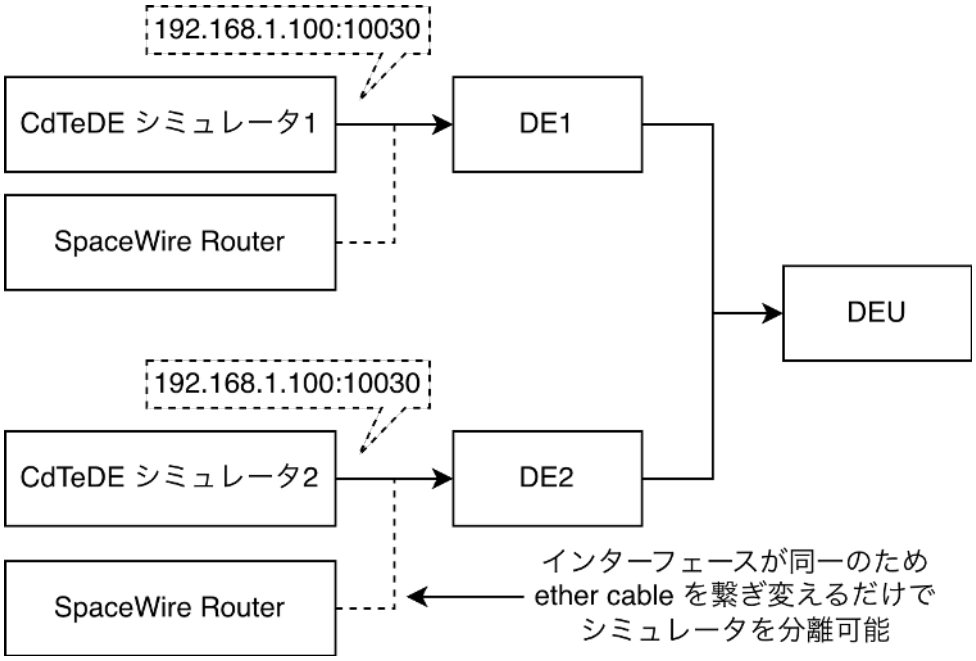


図 4.1: DE シミュレーターの構成

測シナリオに応じたデータを生成することも可能である。

しかし、この構成を実現する上で、大きな課題が存在した。それは、SpaceWire Router の SSDTP2 の TCP 通信プロトコルを解釈できるライブラリが存在しないことである。実績のある SpaceWireRMAPLibrary は SSDTP2 の TCP 通信プロトコルを送信することはできるが、受信することはできない。特に SSDTP2 では timecode パケットのヘッダーが SpaceWire Router 側と、受け手側で異なるため、 TimeCode を mock することができない。そのため、 Appendix D で述べるように、 SSDTP2 の TCP 通信プロトコルを解釈できるライブラリ `spw_rmap`[20] を新たに開発し、DE シミュレータに組み込むこととした。

4.3 模擬データの生成方法

シミュレーター要求 1 を満たすために、内部のシミュレータ自体の設計も重要である。DUA の動作をなるべく忠実に再現し、解析ルーチンの試験にも耐えうるデータを生成する必要がある。特に、Quick Look のプログラムで解析可能な程度にリアルなデータを生成するためには、 CdTe-DSD の応答を模擬したデータを生成する必要がある。

DUA シミュレータではプログラム自体の負荷も考慮する必要がある。DUA の Detector-FPGA では内部のクロックが 100 MHz で動作しており、純粋に 100 MHz で動作するようなプログラムを作成すると、シミュレータの負荷が高くなりすぎる可能性がある。別デバイス上で動作させるとはいえ、シミュレータとして使用するデバイスは Raspberry Pi 4 のような組み込み向けの小型デバイスを想定しているため、負荷が高すぎるとリアルタイムで動作させることができなくなる。そのため、 DE ミュレータでは `write_ptr` の読

み出し時にイベント生成を行い、そのイベントに基づいてデータを生成する方式を採用した。

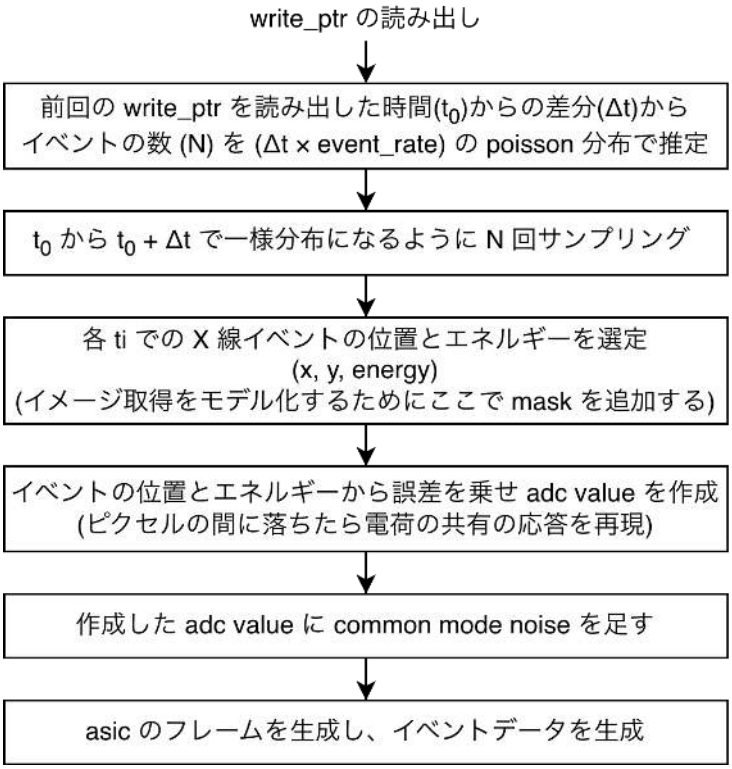


図 4.2: DE シミュレーターのプログラムの流れ

図 4.2 に DE シミュレーターのプログラムの流れを示す。プログラムは write_ptr の読み出しを監視し、 write_ptr が更新された場合にイベントを生成する。イベントは経過時間に基づいて指数分布に従って、Detector-FPGA 内時間 (TI) のリストを生成する。乱数には Mersenne Twister 法を用いており、 C++ 標準ライブラリの `std::mt19937` を使用し、指数分布には `std::poisson_distribution` / `std::uniform_int_distribution` を使用している。経過時間に対して、イベントの発生数を poisson 分布に従って決定し、その数だけ TI リストに uniform 分布に従って TI をソートしながら追加する。

これらの TI リストに基づいて、 DE シミュレータは X 線イベントを生成する。まず各 イベントに対してその位置とエネルギーをランダムに決定する。もし、撮像のテストを行う場合は、特定のパターンに従って位置を分布させることも可能である。その後、決定された位置とエネルギーに基づいて、 CdTe-DSD の応答を模擬する。現在は単純にエネルギーにガウス分布の揺らぎを加えるのみであるが、将来的には CdTe-DSD の応答関数を用いたより精密なシミュレーションを行うことも可能である。

そして最後に、 ASIC ごとの Common Mode 分だけ加算をし、それに一定のガウス分布のノイズを加える。ASIC の Common Mode Noise の計算方法は、 Channel の出力値の中央値を用いるため、ここでも中央値を求めそれを出力データの Common Mode Noise として採用している。それから Header や Footer などを付与し、 SSDTP2 の TCP 通信プロトコルに従ってデータを整形し、 DEU 側に送信する。これらのイベントの生成にお

いては、実データからのランダムサンプリングを用いることも可能であり、より実際の CdTe-DSD の動作に近いデータを生成することもできる。

4.4 実際の運用

本章の最後に、この DE シミュレータの実際の運用手順と実際に生成されたデータを示す。DUA シミュレータを使用する場合は、 SpaceWire Router の代わりに DUA シミュレータを起動し、ethernet ケーブルで DE と接続し、 DUA シミュレータを起動する。あとは 3.4 節の DE の観測手順に従って DE のソフトウェアを起動し、観測を行うだけである。

図 4.3 に DUA シミュレータを使用して作成したイベントのイベントの間隔を横軸に、そのカウントを縦軸にとったヒストグラムを示す。0 から 300 μs までの範囲のカウントがないのは、このシミュレータのデッドタイムが 300 μs に設定されているためである。それ以降の部分は指数分布に従っており、 DUA シミュレータが正しくイベントの間隔を指数分布に従って生成していることが分かる。図 4.4 には DUA シミュレータを使用して作成したイメージを示す。ここでは、任意のパターンに従ってイベントの位置を決定することが可能であることを示すために、鹿のイメージを模擬したイベント位置を生成している。実際には任意のパターンに従ってイベント位置を分布させることが可能であり、このようにイメージングのテストを行うことも可能である。

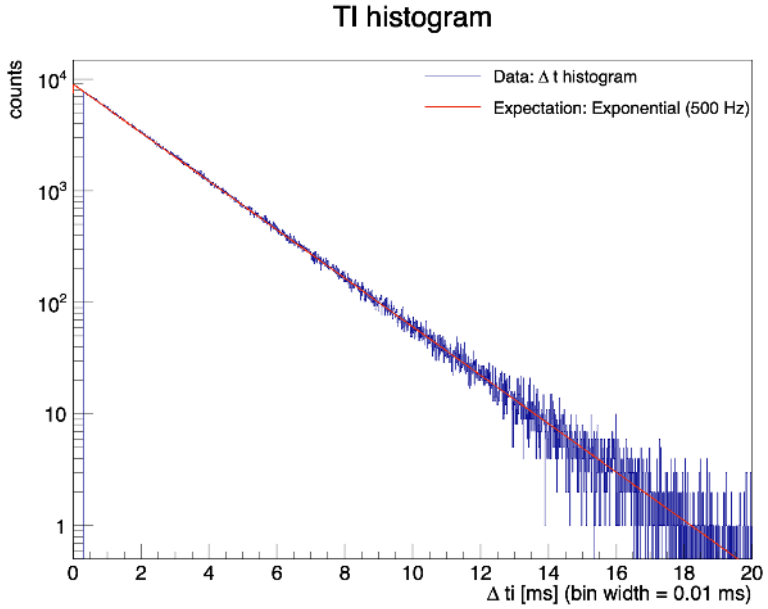


図 4.3: イベント間隔のヒストグラム。 赤線はシミュレーションの設定値 (500 counts/s) に基づいて計算された理論値の指数分布を示す。シミュレーション結果が理論値に良く一致していることが分かる。

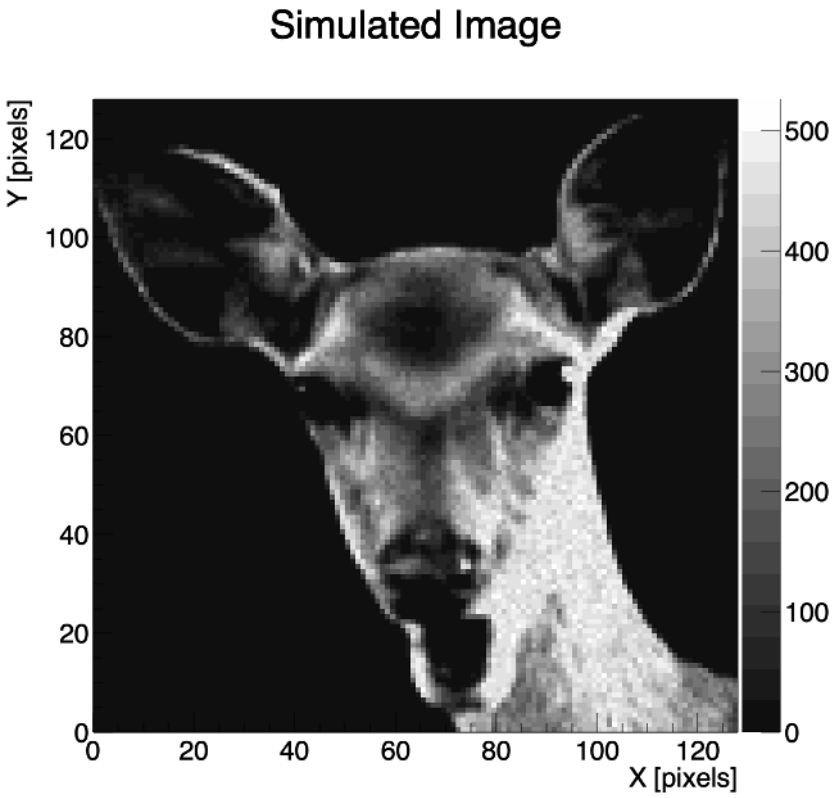


図 4.4: DUA シミュレータを使用して作成したイメージ。colorbar は各ピクセルのカウント数を示す。

4.5　まとめ

本章では DUA シミュレータの開発について述べた。DUA シミュレータは DEU の開発に必須のものであり、DE のコードを一切変更せずに動作させることができるように設計した。さらに、 CdTe-DSD の応答を模擬したリアルなデータを生成することができ、様々な観測シナリオに応じたデータを生成する柔軟性も持つように設計した。これにより、 NASA 側で DEU の設計・動作テストを行うことが可能となった。

第5章　DE の性能評価

この章では DE の性能評価について述べる。DE の性能は、入力段である DUA および SpaceWire 通信系の性能上限により制約されるため、まず DE の性能評価を正しく解釈するための前提として DUA 側の性能評価を行う。5.2 節では DUA の性能評価が DE の性能評価において果たす役割とその重要性について述べ、5.3 節では DUA のハードウェア限界性能の評価について述べる。この節では FPGA ボード SPMU001 を用いて SpaceWire の各種パラメータを変更し、SpaceWire 通信における理論値と実測値を比較することで、DUA のハードウェア限界性能を定量的に評価した結果を示す。 これらの DUA のハードウェア限界性能の評価結果を踏まえ、続く 5.4 節では、実際に DE のソフトウェアと複数の FPGA ボードを用いた DE 全体の性能評価について述べる。そして、これら一連の評価結果に基づき、SuperHERO の気球実験において DE が要求性能を十分に満たすことを示すとともに、将来的な DE の活用においても十分な性能を有することを示す。

5.1　はじめに

DE の性能を決定する要素として、DE 自体の処理能力と、DUA 自体の処理能力がある。ここではまずはじめに、DUA 側の性能、つまり DUA の SpaceWire そして SuperHERO で使用するハードウェアの限界性能について評価を行った結果について述べる。SPMU001 を使用した SpaceWire 通信において制限となる要素を明らかにし、それらが出せる限界の性能について評価し続く章で述べる DE の性能評価の理解に役立てる。

5.2　DUA の性能評価の重要性

DE の性能は DUA の性能に大きく依存する。特に SpaceWire 通信に TCP/IP 通信を使用している DUA のネットワークでは TCP 由来の遅延が発生することが多い。CdTe-FPGA や Detector-FPGA の IP で使用されている SpaceWire はコマンドの buffer 数に制限があるため、非同期に複数のリクエストを送信することができない。もし、非同期に複数のリクエストを送信できる場合、TCP の応答を待つことなく次のリクエストを送信することができるため、 TCP 由来の遅延をある程度吸収することができる。しかし、非同期に複数のリクエストを送信できない場合、 TCP 由来の遅延がそのまま通信全体の遅延に影響を与えてしまう。そのため、 DUA の SpaceWire 通信におけるハードウェアの限界性能を評価することは DE の性能評価と限界性能を理解する上で重要な要素となる。

5.3 DUA のハードウェア限界性能の評価

5.3.1 評価方法

SpaceWire Router を含む DUA + DE のハードウェアの性能を評価するために、二つのセットアップ セットアップ A (図 5.1) セットアップ B (図 5.2) に示すような構成を用意した。Setup A では、Raspberry Pi 4 を Router に Ethernet 接続して Router に接続されている SpW-FPGA に RMAP read コマンドを送信して、FPGA 内部の DDR2 メモリからのデータの読み出しを行う。この構成では SpaceWire を介さないため、SpW-FPGA 及び Detector-FPGA に自体のハードウェアの限界性能と FPGA IP の性能限界を評価することができる。

Setup B では、 DE ボードを Router に Ethernet 接続、さらにもう一台の Detector-FPGA を SpaceWire ケーブルで Router に接続し、SpaceWire 経由で Detector-FPGA の DDR2 メモリからのデータの読み出しを行う。この構成では SpaceWire を介した場合のハードウェアの限界性能を評価することができる。

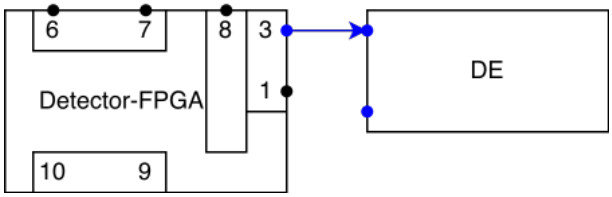


図 5.1: ハードウェア性能評価のセットアップ A

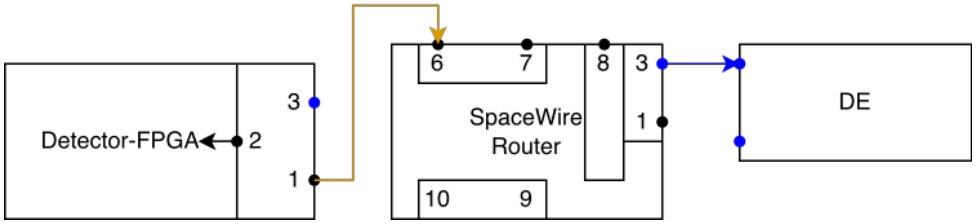


図 5.2: ハードウェア性能評価のセットアップ B

どちらのセットアップにおいても、計測したのは RMAP read コマンドを利用した読み出し速度である。RMAP read コマンドを送信したタイミングから reply パケットを受信し終わるまでの時間を計測し、それを複数のリンクレートで行うことによって、 DUA での SpaceWire 通信におけるハードウェアの限界性能を評価した。リンクレートに関して、 Setup A では SpaceWire を介さないためリンクレートの影響は受けないため、一回の評価のみを行なった。

評価においては RMAP read のみの速度を評価している。これは DE が基本的に「データの読み出し」を主な役割としているためである。また特に RMAP write リクエストに関しては no reply オプションを使用することで reply パケットを受信しない設定にできるため、SpaceWire のネットワークを混雑させることなくリクエストを送信できる方法

も存在し、今回の評価方法である request から reply を受信するまでの時間を計測する方法は RMAP write には適さないためである。

SuperHERO で使用する FPGA では特定のレジスターを変更することで SpaceWire のリンクレートを変更することができる。このレジスターは txck_div と呼ばれ、SpaceWire のクロック周波数を 100/(txck_div + 1) MHz に設定することができる。今回の測定では、txck_div を 9, 4, 2, 1, 0 に設定し、それぞれ 10 MHz, 20 MHz, 33 MHz, 50 MHz, 100 MHz の 5 つのリンクレートで評価を行った。

5.3.2 SpaceWire のリンクレートとデータレートの関係

SpaceWire は双方向に通信データを送る。そのデータレートは使用されるクロックの周波数に依存する。SpaceWire 規格上では、ハードウェアの最低の bitrate は 2 Mbps、最高の bitrate はハードウェアの限界に依存するとされているが、SpaceWire ケーブルは差動信号を採用しており、 8 本の信号線 (Dout+, Dout-, Sout+, Sout-, Din+, Din-, Sin+, Sin-) から構成されている。Data 線 (D+, D-) と Strobe 線 (S+, S-) の 2 組の差動信号を使用してデータを伝達している。クロックは、この Data 線と Storabe 線の XOR によって生成されるため、送信側が一方的にクロックを決定することができる。

データエンコーディングは 1char (8bit) に対して、 control bit と parity bit を付加した 10 bit のデータとして伝送される。従って、理想的なデータレートは以下のように計算される。

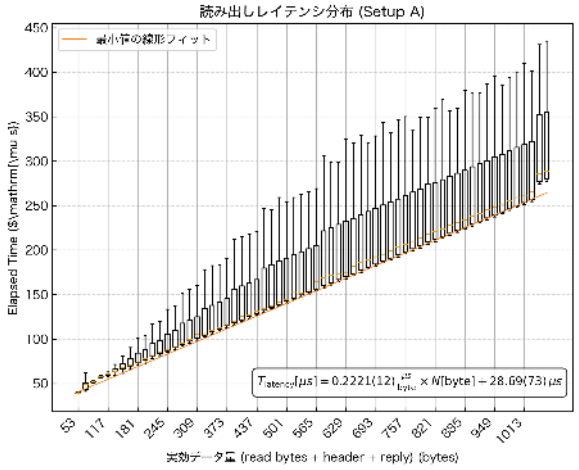
$$R_{data} = \frac{8}{10} \times R_{clock} \tag{5.1}$$

SuperHERO で使用される SpW-FPGA / Detector-FPGA では SpaceWire のクロック周波数は 100/(txck$_{\mathrm{d}}$iv + 1) MHz で設定可能であり、RMAP 上のパケット通信においては最大で 100MHz $\times \frac{8}{10} = 80$Mbps のデータレートが可能である。

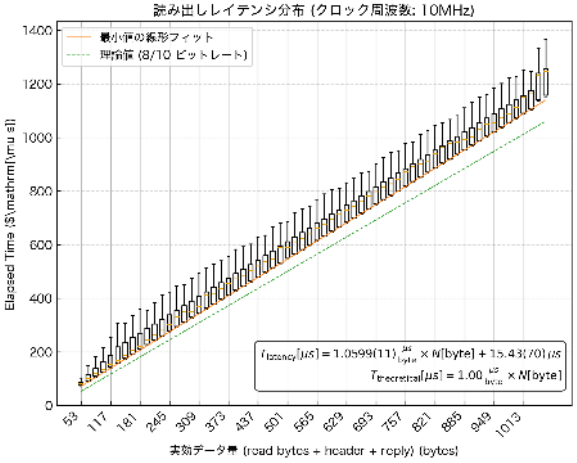
5.3.3 セットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係

図 5.3 にセットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係を示す。横軸には実効データサイズ (bytes) を、縦軸には読み出しにかかる時間 (µs) を示している。RMAP Read では、読み出しデータサイズに加えて RMAP のヘッダー等の追加のデータが付加されるため、実効データサイズは読み出しデータサイズよりも大きくなる。今回の評価では、読み出しデータサイズ + 37$bytes$ が実効データサイズとなる。各 bytesize において 10000 回ずつ評価を行い、それを box plot で示している。

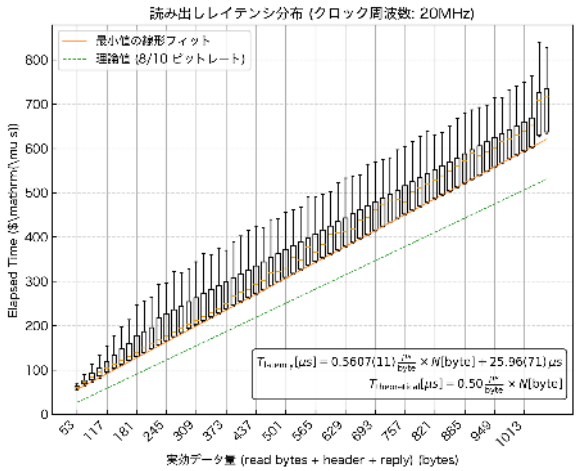
表 5.1 に、最小値を用いて線形フィッティングを行った結果を示す。それぞれのデータ点において、読み出しにかかる時間はばらつきが大きいため、最小値を用いてフィッティングを行うことで、ハードウェアの限界性能に近い値を評価している。全体の傾向として、各リンクレートにおいて読み出しにかかる時間はデータサイズに比例して増加することがわかる。特に、理論値と比較すると若干並行とは異なるが概ね似た傾きに示している



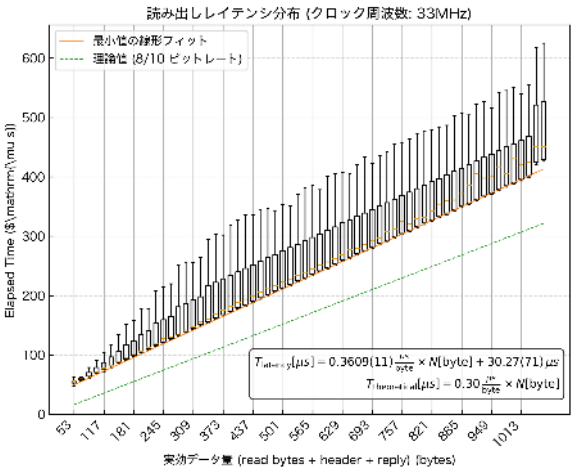
(a) セットアップ A



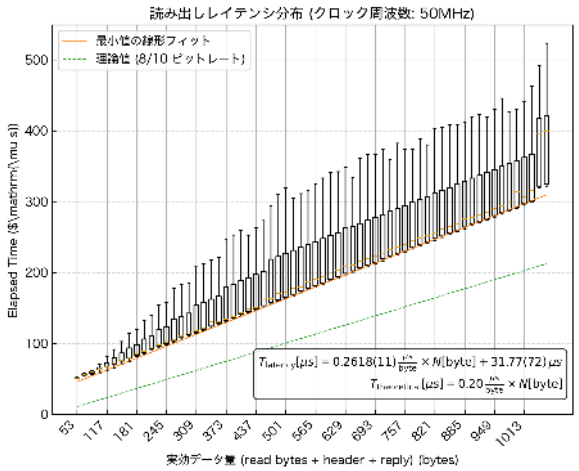
(b) セットアップ B, 10 MHz



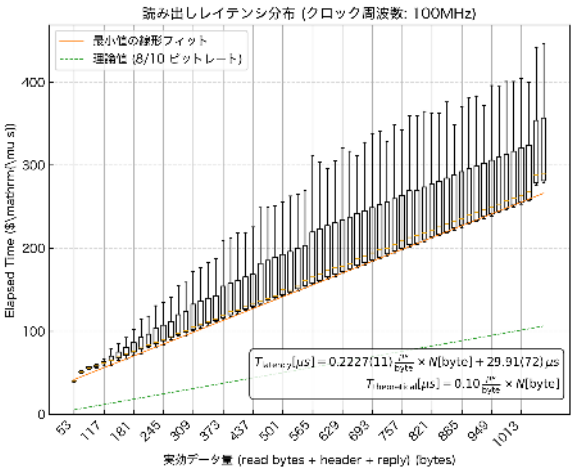
(c) セットアップ B, 20 MHz



(d) セットアップ B, 33 MHz



(e) セットアップ B, 50 MHz



(f) セットアップ B, 100 MHz

図 5.3: セットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係。箱ひげ図は各読み出しデータサイズにおける 10000 回の評価結果を全て用いた分布を示している。フィッティングは最小値を用いて行っている。

ことから、理論値と比べて constant + $\varepsilon \times$ data size の形で読み出しにかかる時間が決定されていることがわかる。

また特に、 1000 bytes を超えたあたりから、データ点がフィッティング結果に対して離れる傾向が見られる。これは、 RMAP read コマンドにおいて 1 パケットあたりの最大データサイズが 1024 bytes に制限されている等の要因が影響している可能性が考えられる。

セットアップ A, B 両方ではリンクレートに関わらず読みだしにかかる時間には 30 μs 程度のオーバーヘッドが存在し、その後にデータサイズに比例した時間がかかることがわかる。セットアップ A, B で特にこのオーバーヘッドに大きな差がなく、誤差の範囲内であることから、SpaceWire 通信に起因するオーバーヘッドはソフトウェア処理や TCP 由来の遅延と比べて無視できる程度であることが示唆される。

表 5.1: セットアップ A, B における読み出しデータサイズと読み出しにかかる時間の関係のフィッティング結果

セットアップ	リンク設定	$a\,[\mu\mathrm{s\,B^{-1}}]$	$b\,[\mu\mathrm{s}]$	$a_{\mathrm{theory}}\,[\mu\mathrm{s\,B^{-1}}]$
セットアップ A	router 非経由	0.2221(12)	28.69(73)	—
セットアップ B	(10 MHz)	1.0599(11)	15.43(70)	1.00
	(20 MHz)	0.5607(11)	25.96(71)	0.50
	(33 MHz)	0.3609(11)	30.27(71)	0.30
	(50 MHz)	0.2618(11)	31.77(72)	0.20
	(100 MHz)	0.2227(11)	29.91(72)	0.10

線形モデル：$T_{\mathrm{latency}}[\mu\mathrm{s}] = a\,[\mu\mathrm{s\,B^{-1}}] \cdot N[\mathrm{B}] + b\,[\mu\mathrm{s}]$.
理論値：$T_{\mathrm{theoretical}}[\mu\mathrm{s}] = a_{\mathrm{theory}}\,[\mu\mathrm{s\,B^{-1}}] \cdot N[\mathrm{B}]$.

5.3.4 小データの読み出しにかかる時間の評価

SuperHERO の DE では、write pointer の読み出しや CdTe-DSD の各種設定処理において、1 word（4 bytes）のデータ読み出しが高頻度で発生する。そのため、4 bytes のデータを読み出す際に要する時間およびその分散を定量的に評価することは、DE の性能評価において重要な指標となる。ここでは、 4 bytes のデータを RMAP read コマンドで読み出す際に要する時間の分布を評価した。

図 5.4 に、SPMU001 を用いた 4 bytes 読み出し処理に要する時間分布を示す。本計測では、4 bytes のデータを読み出す RMAP パケットを送信し、その reply パケットの受信完了までに要した時間を 10,000 回測定した。通信のトップスピードに近い状態での性能を評価するため、計測に先立ち、同一条件で 4 bytes の連続読み出しを 20,000 回実行するウォームアップを行っている。

図 5.4 の横軸は読み出しに要した時間 (µs) を、縦軸は各リンクレートにおける読み出し時間の分布を示している。評価は、10 MHz、20 MHz、33 MHz、50 MHz、100 MHz の 5 種類の SpaceWire リンクレートに対して実施した。また、比較対象として、SpaceWire 通信を

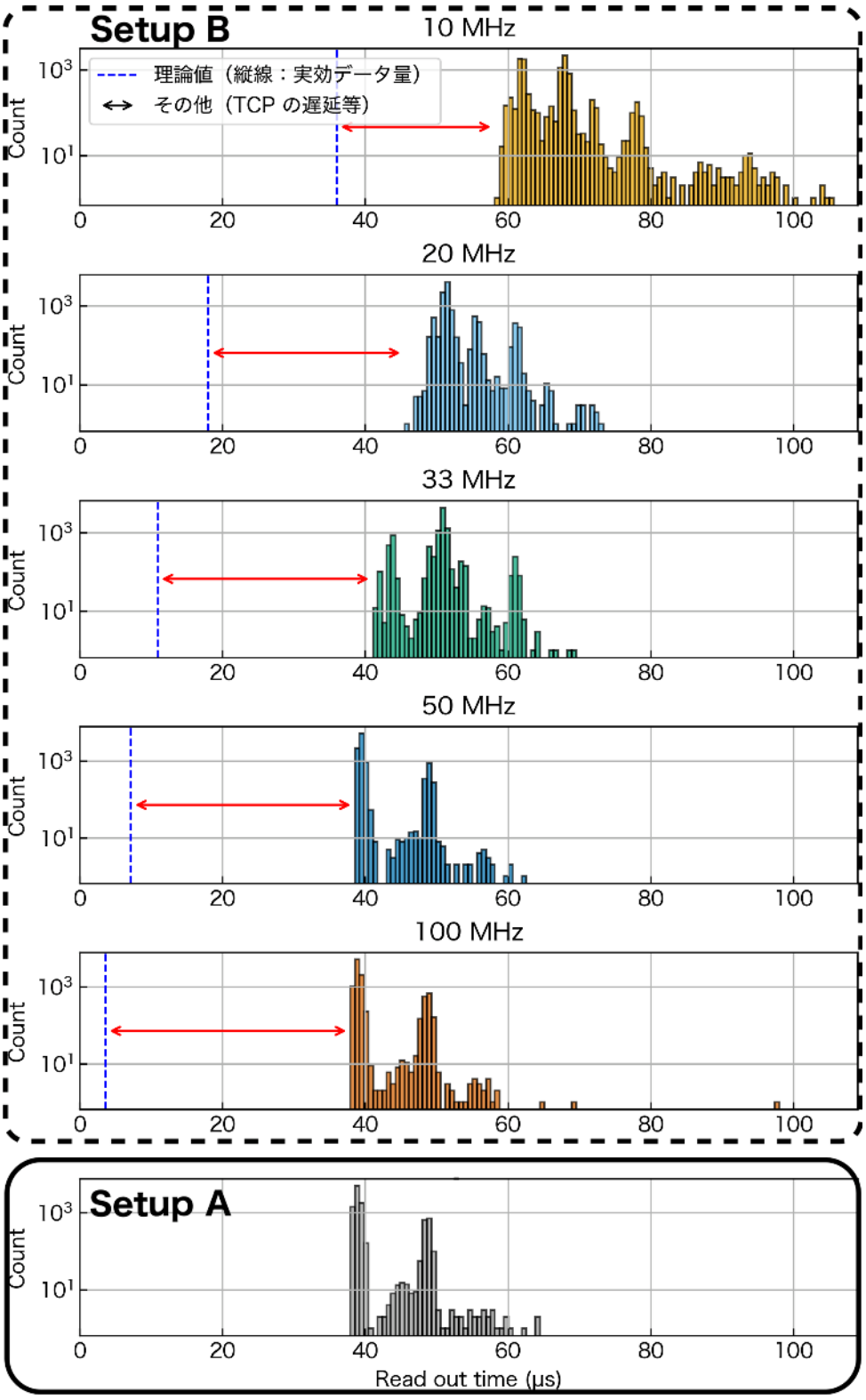


図 5.4: SPMU001 における 4 bytes の読み出しにかかる時間の分布

介さずに SPMU001 内部の DDR2 メモリへ直接アクセスすることで得られた、SPMU001 自体の限界性能も併せて示している。

各リンクレートにおける測定結果は、読み出しデータ量と RMAP ヘッダー長から算出される理論上の最小読み出し時間と比較して、有意に大きな値および分散を示している。この乖離は、SpaceWire 通信における TCP 由来の遅延、および SPMU001 内部の処理遅延が重畳して影響しているためであると考えられる。

特に分散に着目すると、同一リンクレートにおいても最小値と最大値の差が 2 倍以上に達するケースが確認された。TCP 通信はジッターが大きく、通信遅延が確率的に変動する特性を有していることが知られており、SPMU001 を用いた SpaceWire 通信においても、その影響が顕在化していると解釈できる。以上の結果から、4 bytes のような小容量データの読み出しにおいては、リンクレートの向上だけでは遅延分散を十分に抑制できず、通信方式およびデバイス内部処理の影響が支配的であることが示された。

5.3.5 1 イベントフレーム (32KB) の読み出しにかかる時間の評価

DE では、CdTe-DSD から取得したイベントデータを 1 イベントフレーム（32 KB）単位で読み出す。この読み出し方には細かなパケットに分割して送る方法や、一度に大容量のデータを送る方法などが考えられるが、ここでは 1 イベントフレーム（32 KB）のデータを一度に読み出すのに要する時間およびその分散を評価した。一般に、より多くのデータを一度に読み出すことで、通信オーバーヘッドの相対的影響が低減されることが期待される。

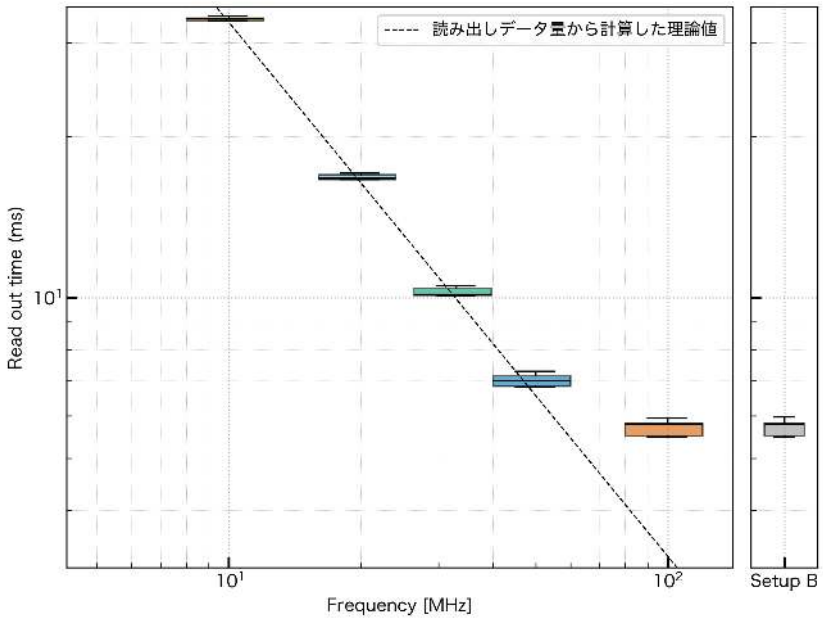


図 5.5: SPMU001 における 32,768 bytes の読み出しにかかる時間の分布

表 5.2: 32 768 bytes 読み出し時のリンクレート別速度比

リンクレート	最小読み出し時間に対する割合	最大読み出し時間に対する割合
10 MHz	99.2 %	97.4 %
20 MHz	98.6 %	95.8 %
33 MHz	98.6 %	94.5 %
50 MHz	96.5 %	90.0 %
100 MHz	59.8 %	55.1 %

SuperHERO の DE では、CdTe-DSD から取得したイベントデータを 1 イベントフレーム（32 KB）単位で読み出す。そのため、1 イベントフレーム（32 KB）のデータを読み出すのに要する時間およびその分散を評価した。一般に、より多くのデータを一度に読み出すことで、通信オーバーヘッドの相対的影響が低減されることが期待される。

図 5.5 は、SPMU001 における 32 KB のデータ読み出しに要する時間分布を、log–log スケールの箱ひげ図として示したものである。本計測では、32 KB のデータを読み出す RMAP パケットを送信し、その reply パケットの受信完了までに要した時間を 10,000 回測定した。通信のトップスピードに近い状態での性能を評価するため、計測に先立ち、4 bytes のデータ読み出しを 20,000 回連続で実行するウォームアップを行っている。

図 5.5 において、横軸は SpaceWire のリンクレート（MHz）を、縦軸は読み出しに要した時間（μs）を表している。10 MHz、20 MHz、33 MHz、50 MHz、100 MHz の 5 種類のリンクレートにおける評価結果に加え、SpaceWire 通信を介さずに SPMU001 内部の DDR2 メモリへ直接アクセスすることで得られた、SPMU001 自体の限界性能も併せて示している。

各リンクレートにおける読み出し時間の分布は、読み出しデータ量および RMAP ヘッダー長から算出される理論上の最小読み出し時間に対して、比較的近い値に集中している。これは、32 KB のような大容量データの読み出しでは、通信オーバーヘッドの影響が相対的に小さくなり、実効性能が理論上の最小読み出し時間に近づくためであると考えられる。

また、リンクレートの増加に伴い読み出しに要する時間が短縮されていることが確認できる一方で、100 MHz では他のリンクレートと比較して改善が頭打ちとなっており、SPMU001 自体の限界性能と同程度の値に収束していることがわかる。

表 5.2 に、各リンクレートにおける最小読み出し時間および最大読み出し時間が、理論上の最小読み出し時間に対して占める割合を示す。最小読み出し時間に着目すると、10 MHz から 50 MHz までは理論値の 96% 以上の性能を達成しているが、100 MHz では理論値の約 60% 程度まで低下している。同様に、最大読み出し時間についても、50 MHz までは理論値の 90% 以上を維持しているのに対し、100 MHz では理論値の約 55% 程度にまで低下していることが確認できる。

5.3.6 まとめ

この節では SPMU001 を使用した SpaceWire 通信におけるハードウェアの限界性能について評価した結果について述べた。SPMU001 の SpaceWire 通信においては非同期通信が使用できないため、TCP 由来の遅延を排除することができず、高頻度での小容量データの読み出しにおいては、遅延が大きく効率を下げる要因となっていることがわかった。

5.3.4 節で述べた 4 bytes の読み出しに要する時間の評価では、リンクレートを向上させた場合においても通信遅延のジッターが大きく、読み出し時間の分散を十分に抑制できないことが確認され、理論上の最小読み出し時間から大きく乖離していることが示された。一方で、1 イベントフレーム (32KB) の読み出しに要する時間の評価では、リンクレートの向上に伴い、読み出し時間が理論上の最小読み出し時間に近づく傾向が確認された。しかしながら、50 MHz を超えるリンクレートでは SPMU001 自体の限界性能に到達し、それ以上の性能向上は得られないことが示し、SPMU001 の現実的なリンクレートの上限は 50 MHz であると結論づけられた。そして、これはそもそもの SPMU001 の処理能力がボトルネックとなっているか、SPMU001 に搭載された DDR2 メモリの転送速度がボトルネックとなっている可能性が考えられる。そのため、将来的に 100 MHz 以上のリンクレートを使用する場合には、SPMU001 自体の処理を見直す必要がある。SuperHERO の気球実験においてこの 50 MHz で十分なレートを確保できることは 5.4 節で述べる DE の性能評価の結果から十分に示される。

5.4 DE の性能評価

DE は最大 7 台の DUA からのデータを受信し、上位電子系へ中継する役割を担うため、その性能は DUA からのデータを安定して受信できるかどうかを決定づける重要な要素である。本節では、DE の性能評価を行い、DE が可能とするデータ転送速度を明らかにする。

5.4.1 実験装置

DE の性能試験では、なるべく SuperHERO で使用される DUA に近い構成を用いることが望ましい。単純な試験の為、以下のような構成で実験を行った。 (図 5.6、図 5.7)

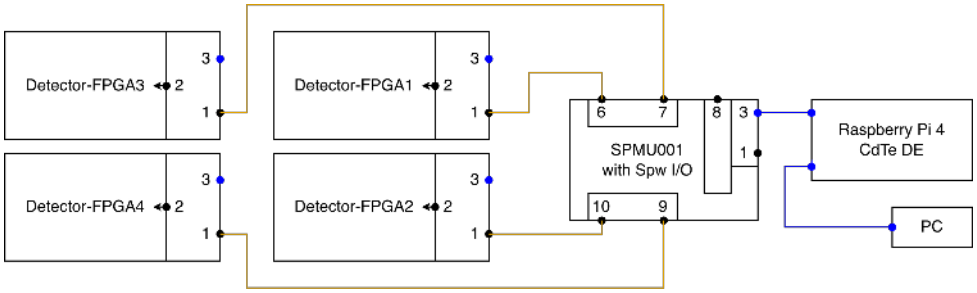


図 5.6: 4 台の DUA と 1 台の DE の接続図

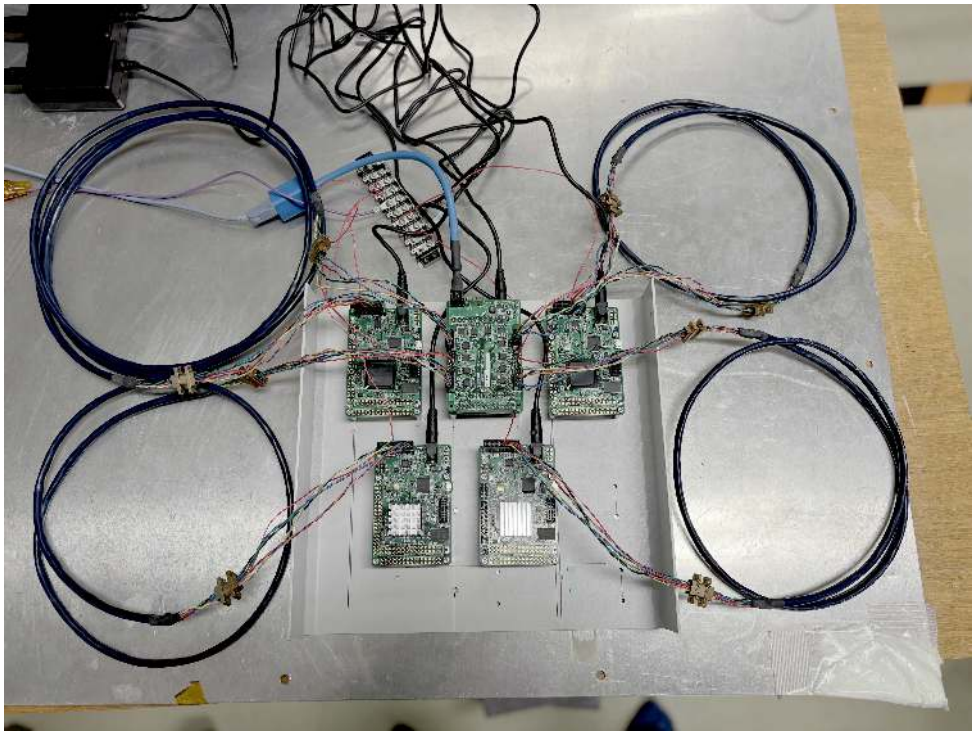


図 5.7: 4 台の Detector-FPGA と 1 台の SpaceWire Router そしてそれに繋がれた DE (Raspberry Pi 4) の実験装置

図 5.6 に示すように、4 台の DUA 相当の Detector-FPGA を SpaceWire Router に SpaceWire ケーブルを通じて接続し、SpaceWire Router から DE (Raspberry Pi 4) には ethernet ケーブルを通じて接続した。図 5.7 はこのセットアップ図に対応する実験装置の写真である。

DE のソフトウェアは 3 章で述べた DE のソフトウェアを Raspberry Pi 4 上で動作をさせる構成とした。DEU 側の処理速度が計測に影響を与えないように、 DEU 側の処理は MacBook 上で動作をさせ、ethernet ケーブルを通じたデータ通信速度が SpaceWire 通信速度よりも十分に高速になるようにした。

DEU 側では、Appendix G で作成した地上実験用のデータ取得ソフトウェアを使用し、DE 側で受信したデータを保存した。そのためここでは、DE 自体のテストと共に地上実験用のデータ取得ソフトウェアの動作確認も兼ねている。

5.4.2 実験方法

実験にあたる疑似データの生成には FPGA 内蔵の疑似イベント生成機能を使用した。FPGA の内部には疑似乱数により、一定のデータレートの指数分布を持つタイミングでトリガーを書け、psuedo event を生成する機能が搭載されている。この機能を使用し、4 台の Detector-FPGA からそれぞれ異なるデータレートでイベントデータを生成し、DE 側でそれらのデータを受信することで、DE の負荷試験を行う。

ただし、 Detector-FPGA には検出器が搭載されていないため、 pseudo event のデータには asic のデータがなく従って 1 イベントあたりのデータサイズは event の header + foot である 7 (words) のみとなる。さらに、 Detector-FPGA の出すことのできる最大の pseudo event rate は 2110 Hz であるため、最大のレートで出しても 1 秒あたりのデータ量は $2110 \times 7 \times 4 = 59,080$ bytes/s となり、DUA の限界性能の 1/100 にも見たない。そこで、Detector-FPGA の内部にある イベントフレーム切り変えまでの閾値を調整することで 1 フレーム (32 KB) に格納されるイベント数を減らし、結果として見掛けのイベントレートを上げることができる。フレームの大部分は 0 埋めされた状態とはなるが、DE 側は 32KB を一単位としたイベントフレームを扱うため、問題はない。

例えば、 1000 bytes に設定した場合、1 フレームあたりに格納されるイベント数は $\lfloor \frac{1000}{7\times4} \rfloor = 35$ events となり、見掛けの最大のデータレートは $32\ \mathrm{KiB} \times \frac{2110\ \mathrm{Hz}}{35 events} = 1.93\ \mathrm{MiB}/s$ となる。この方法を使用し、4 台の Detector-FPGA からそれぞれ異なる見掛けのイベントレートでデータを生成し、DE 側でそれらのデータを受信することで、DE の負荷試験を行なった。

5.4.3 実験結果

図 5.8 に DE の負荷試験の結果を示す。横軸は 4 台の Detector-FPGA 内で生成された見掛けのデータレートを示し、縦軸は DE 側で実際に受信できたデータレートを示している。DE 側で受信できたデータレートは 4 台の Detector-FPGA 内で生成された見掛けのデータレートの合計に対してほぼ線形に増加し、ある限界値に達した後、飽和することがわかる。

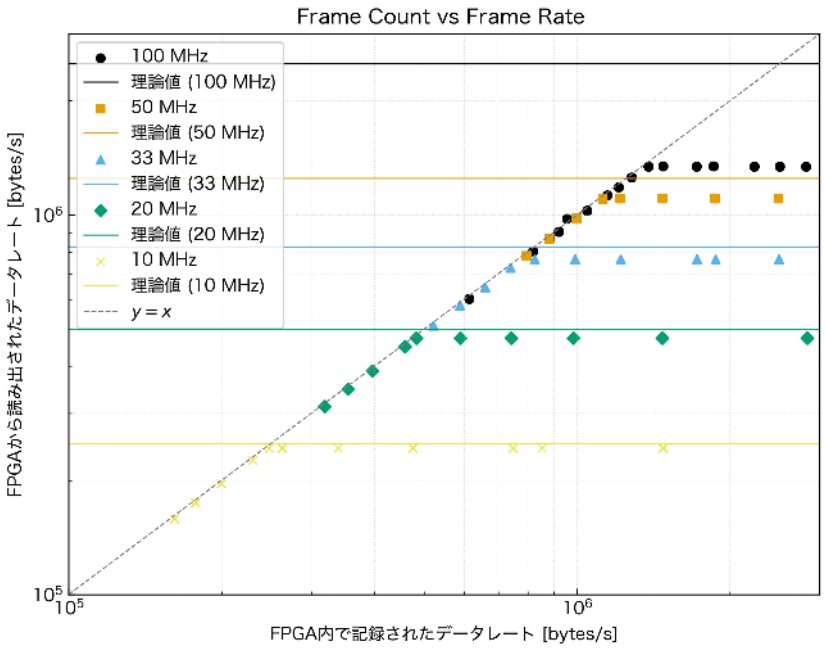


図 5.8: 4 台の Detector-FPGA からのデータを受信する DE の負荷試験結果。4 台間でデータ点が重なっており、図中の点それぞれに 4 台分のデータが対応している。横軸は 4 台の Detector-FPGA 内で生成されているデータのデータレートを示し、縦軸は DE 側で実際に受信できたデータレートを示している。リンクレートの低いものから順にあるデータレートで飽和する様子が確認できる。

図内のデータ点は 4 台の Detector-FPGA からのデータをそれぞれプロットしているが、全ての点でピッタリと重なっている。これは、 DE 側でのデータ受信が 4 台の Detector-FPGA からのデータを均等に処理できていることを示しており、これは DE の測定において公平に各検出器からのデータを処理できることを示している。また、DE 側で受信できたデータレートは最大で約 4.50 MB/s に達していることがわかる。この値は、前章で示した DUA 自体の限界性能である 4.83 MB/s に対して約 93% に達する高い値である。さらに、 SpaceWire 規格が理論的に許容する最大データレートである 5.0 MB/s に対しても約 90% に達する高い値である。以上より、 DE は DUA 自体の限界性能に近い高いデータ転送速度を実現できていることを実証した。

5.4.4 CdTe-DSD のデッドタイム実測と最大イベントレート

検出器の最大のイベントレートはデッドタイムに依存する。CdTe-DSD のデッドタイムは大まかに ASIC 自体の信号処理等に起因する部分と ASIC からのデータ読み出しに起因する部分に分けられる。そのため、何 channel 分のデータを読み出すかによってデッドタイムは変化する。SuperHERO で使用する CdTe-DSD では、スパースモードでトリガー threshold を超えたチャンネルのみを読み出す場合の、デッドタイムの実測は 約 330 μs 程度である。一方、全チャンネルを読み出し 256 チャンネル分のデータを読み出す場合のデッドタイムの実測は 約 1,850 μs 程度である。

CdTe-DSD のデッドタイムは非麻痺型である。一般に、非麻痺型のシステムにおける最大イベントレート R_{max} はデッドタイム τ を用いて以下のように表される。

$$R_{max} = \frac{1}{\tau} \tag{5.2}$$

したがって、スパースモードでは平均デッドタイム $330\mu s$ から CdTe-DSD の最大イベントレートは $\frac{1}{330\times10^{-6}} = 3,030$ events/s 程度であると推定される。同様に、全チャンネル読み出しでは $\frac{1}{1,850\times10^{-6}} = 540$ events/s 程度であると推定される。これらはいずれも SuperHERO で想定される最大イベントレート (2.2) よりも遥かに高い値であり、CdTe-DSD 自体が SuperHERO の観測に十分な性能を有しているとともに、このレートが DE の性能評価における基準となることを示している。

5.4.5 DE の最大処理能力と CdTe-DSD 最大イベントレートの比較

CdTe-DSD のデータはイベントフレーム単位で DE に送信される。そのため、DUA からのデータレートはイベントレートに純粋に比例するわけではない。ここでは、2.5.2 節で示した CdTe-DSD のデータフォーマットを基に、1 フレームあたりのイベント数を見積もり、それを基に DE が処理可能な最大イベントレートを推定する。

1 フレームあたりのイベント数

SuperHERO で使用する CdTeDSD では 4 枚の ASIC が搭載されているため、全チャンネル読み出し時には 1 イベントあたり最大で $4 \times 92 = 368$ bytes のデータが生成される。それに加えて、イベント自体の header (4 bytes) と HK data (20 bytes) そして header (4 bytes) が付加されるため、合計で 396 bytes のデータが一イベントあたり生成される。

FPGA 内では 65,536 bytes のブロックメモリの前半後半を 32,768 bytes ずつ使用し、ダブルバッファとして使用している。そのため、一つの frame は 32,768 bytes から構成されるが、実際にはマージンを取って 32,000 bytes を越えるとバッファを切り替えるようにしている。したがって、全チャンネル読み出し時には 1 フレームあたり最大で $\lfloor \frac{31,999}{396} \rfloor = 80$ イベントを格納することができる。

一方、スパースモードでは ASIC から読み出されるチャンネル数が可変である。一般的には表面と裏面で各 1 チャンネル、合計 2 チャンネルが読み出され、チャージシェアリングが発生した場合にはさらに数チャンネルが読み出される。ここでは典型的な値として読み出しチャンネル数を $N = 3$ と仮定する。このとき、1 イベントあたりのデータサイズは $10 \times 3 + 91 = 121$ bits となり、4 bytes アラインメントすると 32 bytes となる。したがって、1 イベントあたり $32 * 4 + 4 + 20 + 4 = 156$ bytes のデータが生成される。同様に 32,000 bytes を越えるとバッファを切り替えるため、1 フレームあたり最大で $\lfloor \frac{31,999}{156} \rfloor = 204$ イベントを格納することができる。

DE が処理可能な最大イベントレート

DUA 自体の性能評価では、リンクレート 50 MHz において最大 4.83 MiB/s のデータレートを処理可能であることが示されている。DE の性能評価においても、同一条件下で 4.50 MiB/s のデータレートを安定して処理可能であることが確認された。この値は、SpaceWire 規格が理論的に許容する最大データレートの約 90% に相当し、さらに、実測した DUA の限界性能に対しても約 93% に達する。

この結果から、一秒あたりのイベントフレーム数は

$$\frac{4.50\mathrm{MiB/s}}{32\mathrm{KiB}} \simeq 144 \text{ frames/s} \tag{5.3}$$

である。これに 1 フレームあたりのイベント数を掛け合わせると、DE が処理可能な最大イベントレートは、全チャンネル読み出し時に $11,520$ events/s、スパースモード時に $29,376$ events/s と推定される。

以上より、CdTe-DSD が物理的に出力し得る最大イベントレートはデッドタイムによって制限されており、DE の最大処理能力はその制限に対してスパースモードでは $\frac{29,376}{3,030} \simeq 9.7$、全チャンネル読み出しでは $\frac{11,520}{540} \simeq 21.3$ と十分な余裕を有していることが示される。

5.5 まとめ

本章では、DUA のハードウェア限界性能評価の結果を基に、それらが実際の DE の性能にどのように反映されるかを明らかにするため、Detector-FPGA を用いた DE の性能評価を行った。

DUA のハードウェア限界性能評価から、DUA において使用されている FPGA は、設定可能なリンクレートの上限において FPGA 自体の処理性能がボトルネックとなり、SpaceWire 通信規格が規定する理論上の最大データ転送速度には到達しないことが明らかとなった。また、4 bytes の読み出しに要する時間の評価では、リンクレートを向上させた場合においても通信遅延のジッターが大きく、読み出し時間の分散を十分に抑制できないことが確認された。一方で、大容量データ（32 KB）の読み出しに要する時間の評価では、リンクレートの向上に伴い、読み出し時間が理論上の最小読み出し時間に近づく傾向が確認された。しかしながら、50 MHz を超えるリンクレートでは FPGA 自体の限界性能に到達し、それ以上の性能向上は得られないことが示された。以上より、DUA における設定値として最適なリンクレートは 50 MHz であると結論づけられる。

さらに、4 bytes および 32 KB の読み出し性能評価結果を比較すると、DUA の SpaceWire ネットワークでは、小容量データを断続的に読み出すよりも、大容量データをまとめて読み出す方が通信効率の観点から有利であることが明らかとなった。この結果から、FPGA の設定上のリンクレート上限としては 50 MHz で十分であり、その条件下において、SpaceWire 通信が理論的に許容するデータレートの約 90% に相当する高いデータ転送速度を実現できることが示された。

最後に、 DE の性能評価では、4 台の Detector-FPGA からのデータを同時に受信する負荷試験を行い、DE が最大で約 4.50 MiB/s のデータレートを安定して処理可能であることが示された。この値は、DUA 自体の限界性能である 4.83 MiB/s に対して約 93% に達する高い値である。さらに、SpaceWire 規格が理論的に許容する最大データレートである 5.0 MiB/s に対しても約 90% に達する高い値である。これからイベント数を推定すると、全チャンネル読み出し時には最大で約 11,520 events/s、スパースモード時には最大で約 29,376 events/s のイベントレートを処理可能であると推定される。デッドタイムの実測結果から、CdTe-DSD のスパースモードにおけるデッドタイムは平均で 330μs 程度であり、最大イベントレートは $\sim 3,030$ events/s 程度であると推定される。この最大イベントレートで動作していたとしても、 DE は約 9.7 倍の余裕を持っており、2028 年の観測ミッションにおいて 7 台全部が同時に最大レートで動作した場合でも十分な性能を有していることが示された。これらは、 SuperHERO で想定される最大イベントレート (2.2) よりも遥かに高い値であり、DE が 2028 年の観測ミッションにおいて十分な性能を有していることが示された。

第6章　Si-DSD を用いた長時間動作試験

この章では、Si-DSD を用いた長時間動作試験の結果について示す。本試験は、第 5 章までに評価した DE の性能が、実運用に近い長時間連続動作においても維持されることを検証することを目的としている。6.1 節では長時間動作試験が 2028 年の SuperHERO 気球実験において必要である理由について述べ、6.2 節では本試験の目的について述べる。6.3 節では実験方法と実験装置について述べ、6.4 節では実験結果について示す。これらの結果を踏まえ 6.4.1、6.4.2 節では Si-DSD を用いた長時間動作試験において 15 時間にわたり安定してデータが欠けることなく安定して取得できたことを示す。

6.1　長時間動作試験の必要性

2028 年の SuperHERO 気球実験では、CdTe-DSD を用いて約 2 日のフライト中に最長半日程度の連続観測が予定されており、DE は半日以上にわたり耐え間なくデータを取得し続けることが求められる。また、SuperHERO に限らず、観測装置は数週間にわたり安定して動作することが求められ、観測装置とそのデータ収集システムは長時間にわたり安定して動作することが必須である。特に、データの欠損を生じさせないことは重要であり、SuperHERO においても、長時間にわたりデータを欠損することなく安定してデータを取得し続けられるソフトウェアはなくてはならないものである。本章では、Si-DSD [21] を用いた長時間動作試験の結果について示す。

6.2　実験目的

本章では、Si-DSD を用いた長時間動作試験の結果について示す。Si-DSD は SuperHERO で使用される CdTe-DSD と全く同じ ASIC を搭載しており、同じように動作を行うことのできる検出器である。そのため、Si-DSD を用いた長時間動作試験を行うことで、SuperHERO で使用される CdTe-DSD の長時間動作特性を評価することができる。特に、SuperHERO では長時間にわたり安定してデータを取得することが求められるため、Si-DSD を用いた長時間動作試験を通じてその特性を評価することは重要である。さらに、この試験では Si-DSD のイメージング能力も評価することを目的としている。

SuperHERO で使用される CdTe-DSD はミラーと組み合わせて撮像用途に使用される。データが正しく取得できているかどうかを評価するにあたっても、イメージングを通して評価することは重要であり、DUA の SpaceWire 通信や DE のソフトウェアでの安定動作を評価する上でも有用である。

6.3　実験方法/実験装置

本測定においては、SuperHERO で使用する DE の負荷を再現するために、Si-DSD の他性能評価試験で使用した Detector-FPGA を四台使用し、計五台の Detector-FPGA からデータを取得する構成を採用した。SuperHERO で同時使用される DUA は 3 / 4 / 7 台のいずれかである。しかし、ソフトウェアでは特に DUA の台数に依存した処理は行われていないため、本試験では使用可能な Detector-FPGA を最大限使用し、考えられる限りの高負荷環境を再現した。

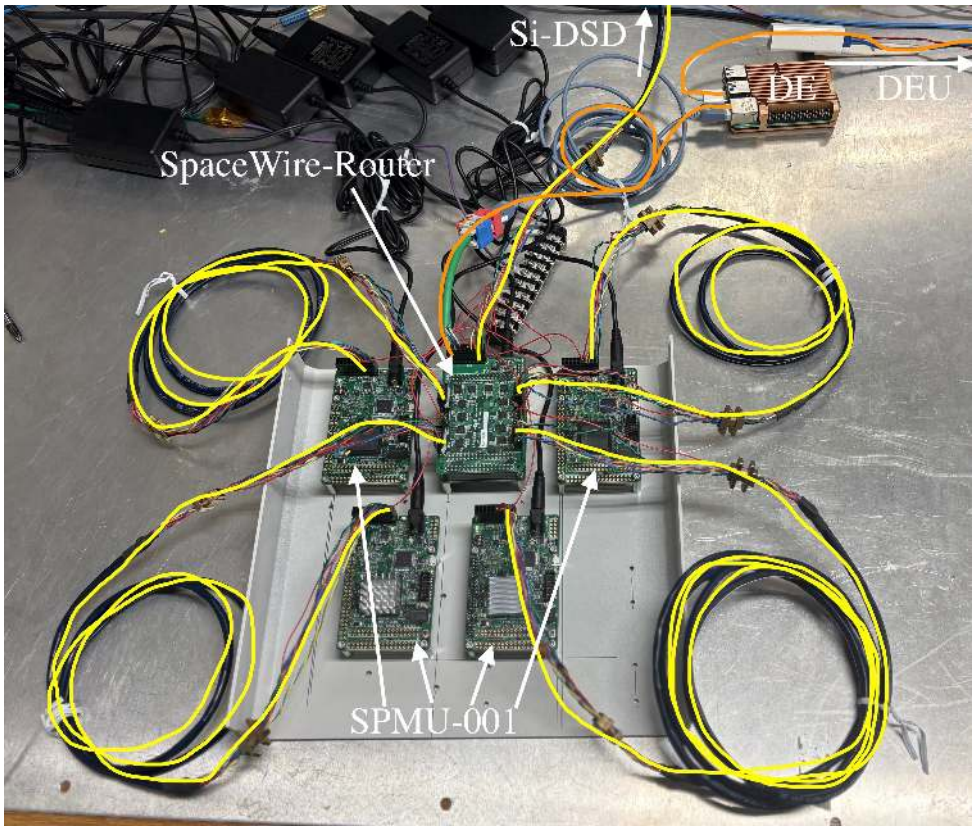


図 6.1: Si-DSD を用いた長時間動作試験の実験装置

図 6.1 は Si-DSD を用いた長時間動作試験の実験装置であり、これは、図 6.2 に示す構成図に対応している。SpaceWire Router のポート 6/7/9/10 に Detector-FPGA を 4 台接続し、ポート 1 には Detector-FPGA を接続した。この Detector-FPGA には Si-DSD が接続されているが、Si-DSD の内部構成は同じであり CdTe-DSD を動かしている FPGA の IP でも動作させることができる。Si-DSD は 図 6.3 に示すように恒温槽内部に配置されている。

図 6.3 に示すように、恒温槽上部には窓が設けられており、検出器の観測面に垂直になるように窓が配置されている。図 6.4 に示す窓に ^{241}Am 線源を取り付け、 Si-DSD の観測面に向けて線源からの X 線を照射した。

図 6.6 に Si-DSD 検出器ユニットの内部を示す。Si-DSD は CdTe-DSD と同じ ASIC を搭載しており、その中身も CdTe-DSD と同じ構成となっている。長時間動作試験では、

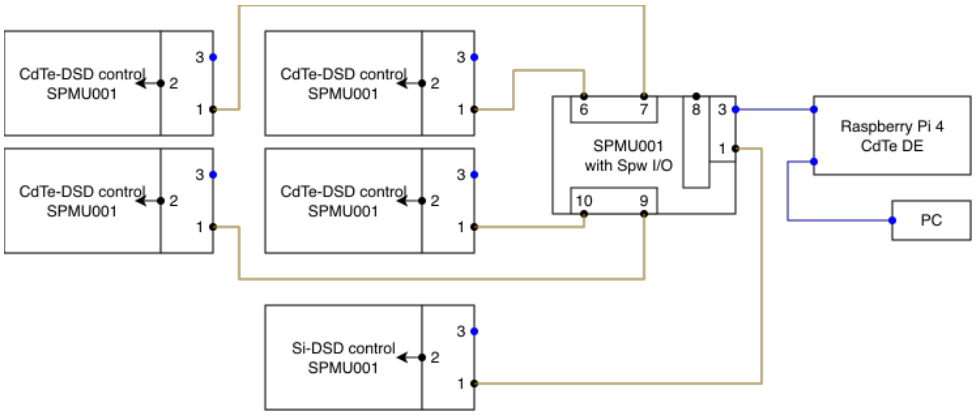


図 6.2: Si-DSD を用いた長時間動作試験のセットアップ

Si-DSD の観測面と一致するように、金属のしおりを配置し、 X 線イメージングを行った。図 6.5 は Si-DSD 上に配置された金属製のしおりを示す。金属製のしおりは、0.5 mm 厚の金属板であり、Si-DSD 上の観測面に密着させて配置している。Si-DSD の観測面にはアルミ蒸着した薄い保護膜が存在し、6.7 に示すように Si-DSD 検出器ユニット全体は金属製の箱に収められている。

図 6.3: 恒温槽内部に配置された Si-DSD

性能試験では Si-DSD を恒温槽内で -25 ℃ に冷却し、 ^{241}Am 線源を用いて X 線を照射した。Si-DSD のカウントレートは約 400 counts/s 程度である。本性能試験では、なるべく SuperHERO での CdTe-DSD の使用条件に近づけるために、5 台からのレートを同等になるように、Detector-FPGA 内部のレジスターを設定し、DE はダミーのデータ

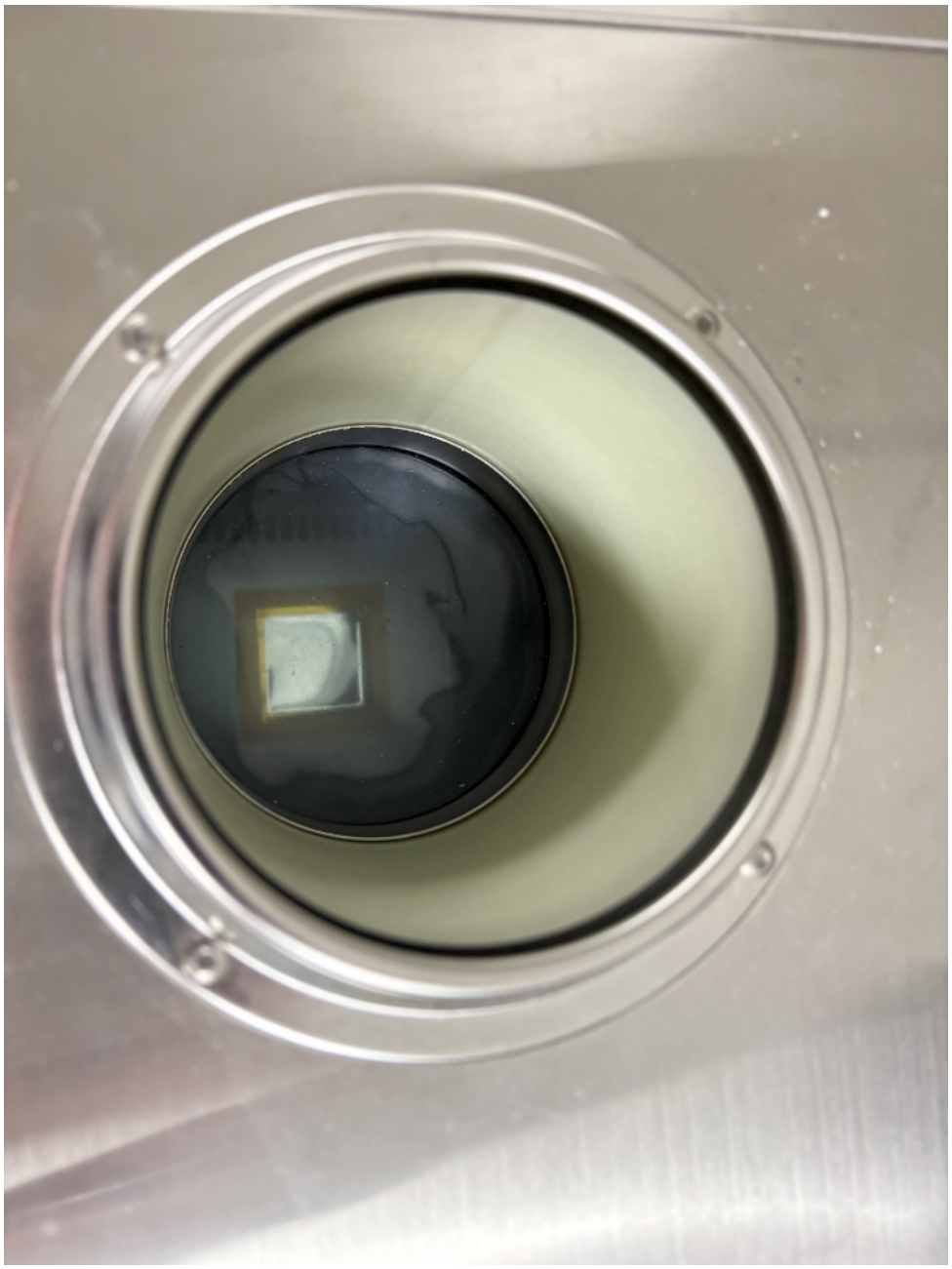


図 6.4: 恒温槽上部に設けられた窓

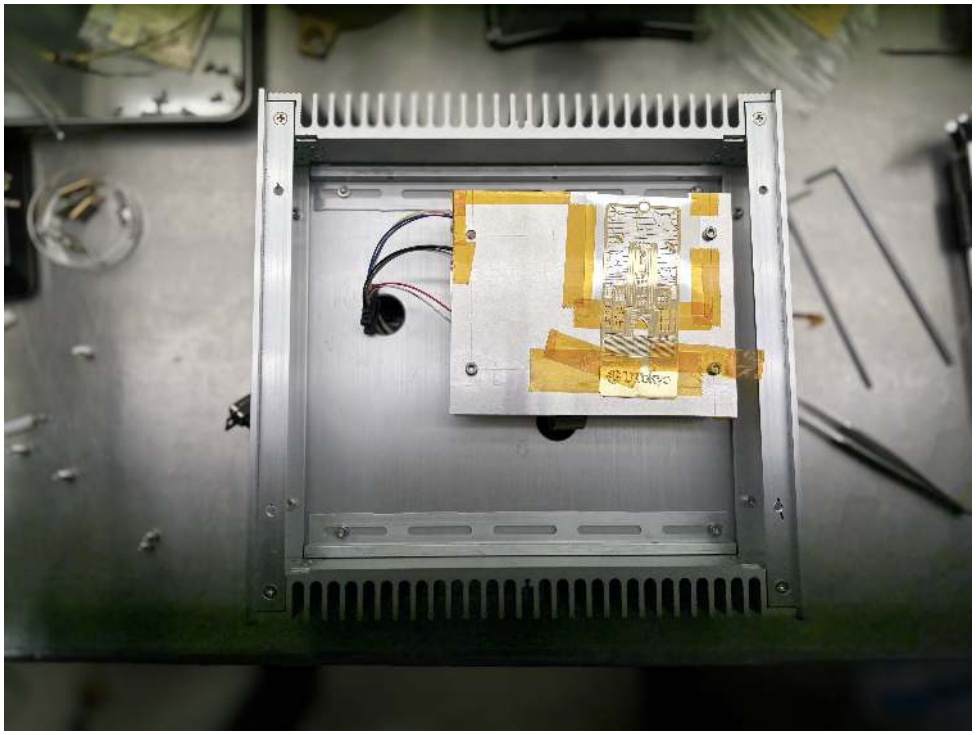


図 6.5: Si-DSD 上に配置された金属製のしおり

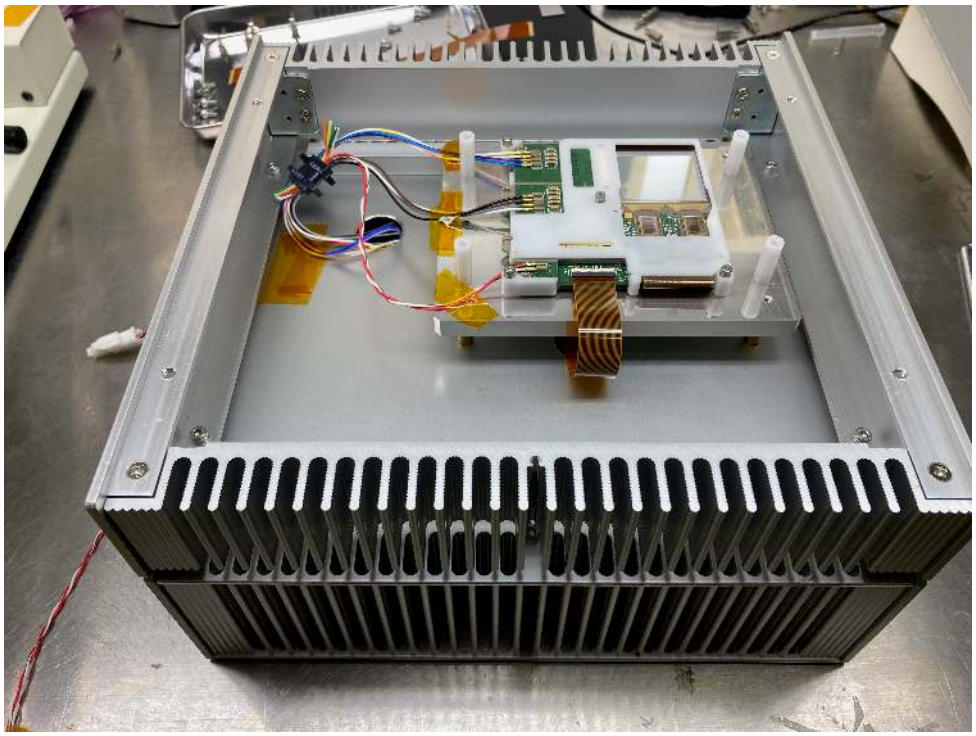


図 6.6: Si-DSD 検出器ユニットの内部

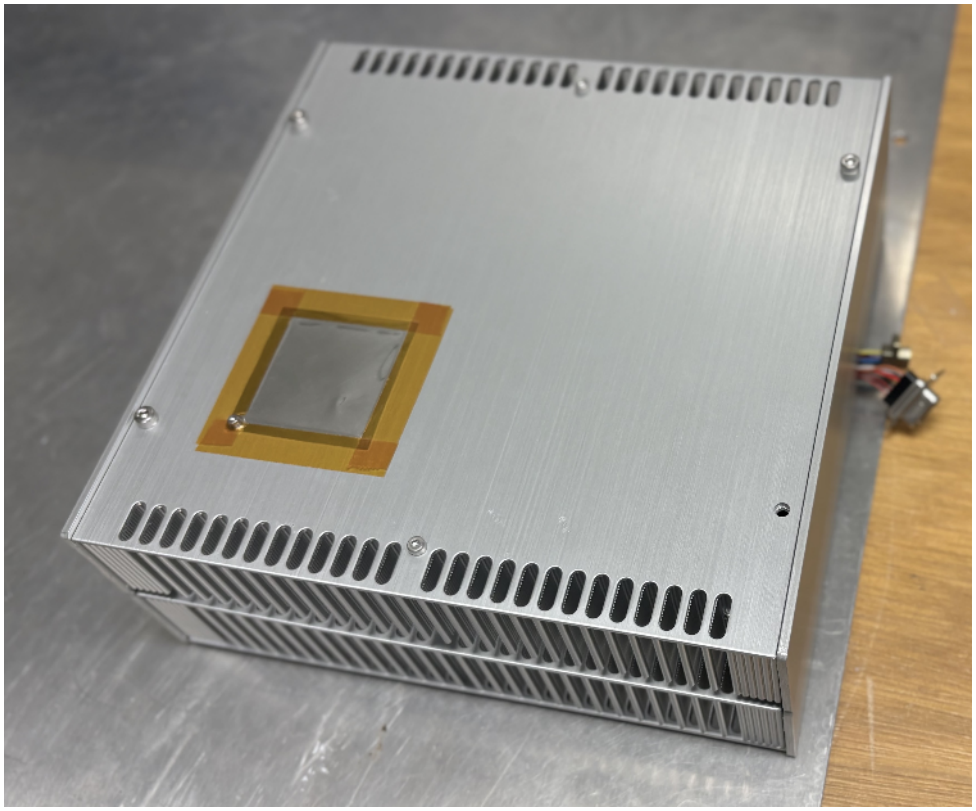


図 6.7: Si-DSD 検出器ユニットの外観

4 台分と Si-DSD からのデータ 1 台分を合わせた、計 5 台分のデータを受信する構成とした。各 Detector-FPGA からのデータレートはイベント数で約 400 events/s を超えるレートとなっており、SuperHERO の気球実験で予想される 約 2.7 counts/s (2.2) を大きく上回る高負荷環境を再現している。その上で、データ取得を絶え間なく 15 時間連続動作させ、長時間にわたる安定動作を評価した。

6.4　実験結果

6.4.1　取得データの健全性

本試験では、Si-DSD を用いた長時間動作試験において、15 時間にわたり安定してデータを取得できることを確認した。以下に、取得したデータの健全性について示す。

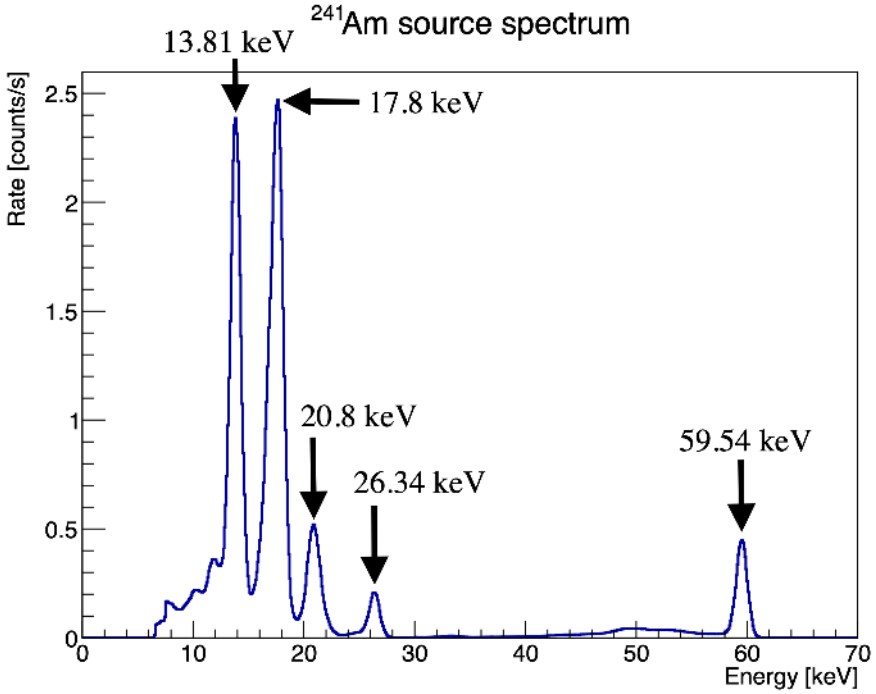


図 6.8: Si-DSD を用いた長時間動作試験で取得したスペクトル

図 6.8 では Si-DSD を用いた長時間動作試験で取得したスペクトルを示す。Si-DSD では CdTe-DSD と同じく両面ストリップ検出器であるため、ground 側と high voltage 側の両方のチャンネルからスペクトルを取得することができる。ここでは、 ground 側のチャンネルからのデータのみを用いてスペクトルを作成している。ground 側を担当する ASIC は 2 台存在し、それぞれエネルギーに対する応答が異なるため、簡易的に 2 次関数でエネルギー較正を行い、両方の ASIC のデータを合わせてスペクトルを作成している (詳細は Appendix H 節を参照)。^{241}Am の 13.81 keV、 17.8 keV、 20.8 keV、 26.34 keV、 59.54 keV のピークが確認できる。

最後の図 6.9 は Si-DSD を用いた長時間動作試験で取得したイメージングである。Si-DSD 上に配置された金属製のしおりの形状が鮮明に確認でき、Si-DSD のイメージング能力が十分であることを示している。このイメージは 11 keV から 23 keV の範囲の X 線イベントを用いて作成している。検出範囲の境界から 2 pixel 分はノイズが多く含まれるため、その部分を除去してイメージングを作成している。

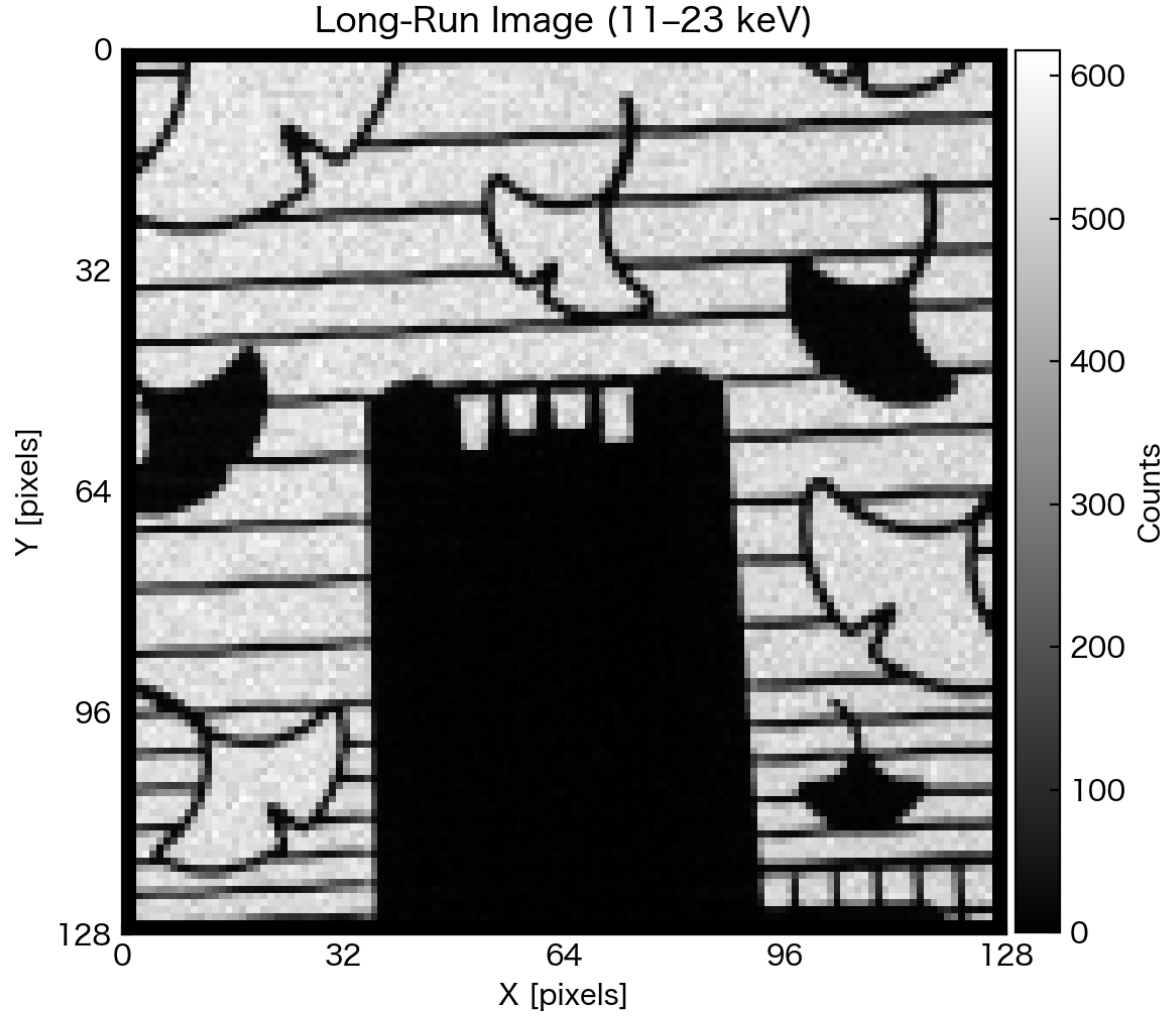


図 6.9: Si-DSD を用いた長時間動作試験で取得したイメージ

6.4.2　長期安定性の検証結果

ここでは、 Si-DSD を用いた長時間動作試験で取得したデータの長期安定性について示す。主にデータに抜けがないかという点と、システム全体としての安定性について評価を行った。

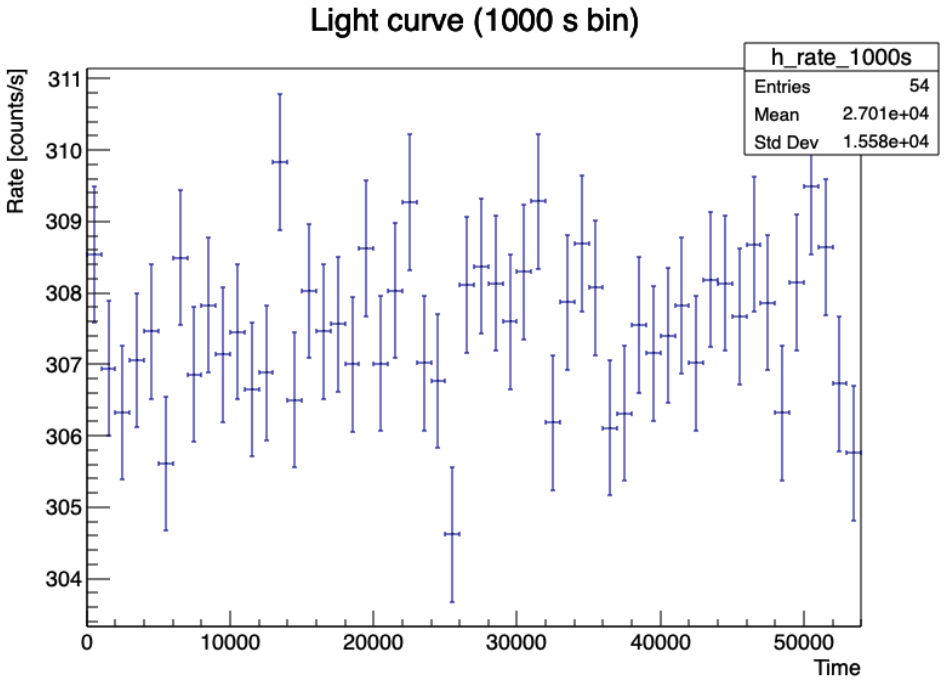


図 6.10: Si-DSD を用いた長時間動作試験で取得したライトカーブ

図 6.10 に Si-DSD を用いた長時間動作試験で取得したライトカーブを示す。各データ点にはイベントをポアソン分布であると仮定した場合の 1σ のエラーバーを付与している。概ね取得イベントは安定しており、15 時間にわたり大きな変動なくデータを取得できていることが確認できた。ライトカーブの平均カウントレートは約 308 counts/s ($\pm$3 counts/s) で安定している。このライトカーブは、実際の X 線イベントを抜きだしたカウントレートを全体の live time で割ったものであり、6.4.1 節で示したスペクトルのイベントを用いて作成している。

図 6.11 に Si-DSD を用いた長時間動作試験で取得したイベント間隔のヒストグラムを示す。これは Pseudo イベントも含めた全イベントのイベント間隔を示している。横軸はイベント間隔を示し、単位は μs である。縦軸はイベント数を示している。赤の点線は指数関数によるフィッティング結果を示しており、イベント間隔が指数分布に従っていることが確認できる。このヒストグラムには、全イベントが含まれており、ti の差分が 14ms を超えるイベントは存在しないことが確認できている。もし、何らかの理由でデータが欠損していたり、システム自体が停止していた期間が存在した場合、ti の連続性が失われ、ti の差分が大きな値を取るイベントが存在するはずである。そのようなイベントが存在しないことから、 15 時間にわたりデータが欠損することなく安定してデータを取得できていることが確認できた。

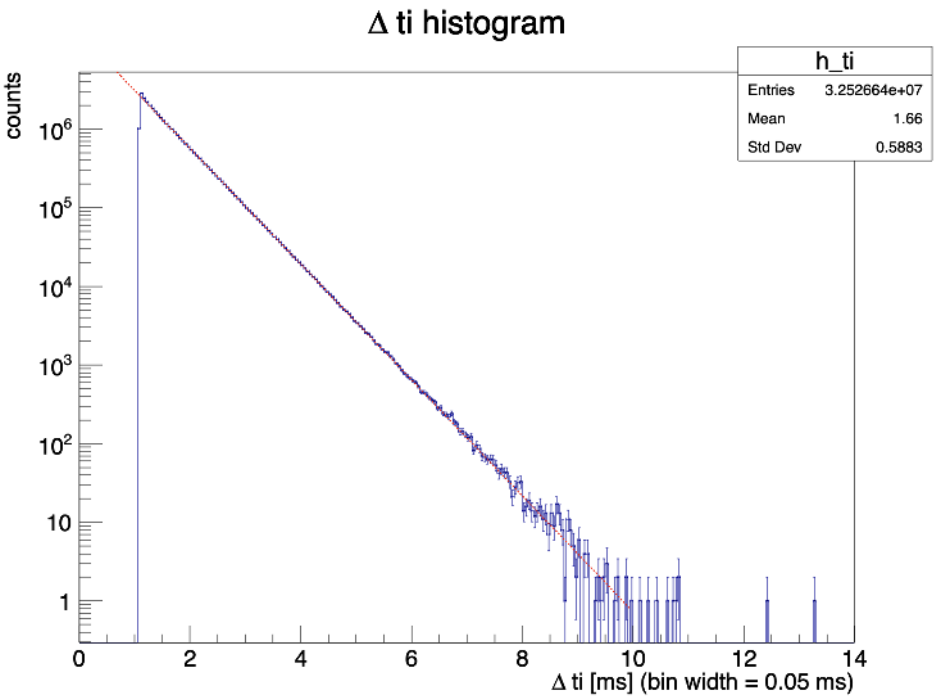


図 6.11: Si-DSD を用いた長時間動作試験で取得したイベント間隔のヒストグラム。Pseudo イベントも含めた全イベントのイベント間隔がこのヒストグラムに示されている。ti の差分が 14ms を超えるイベントは存在しないことが確認できている。

このヒストグラムに対して指数関数フィッティングを行った結果を以下に示す。まず、待ち時間 Δt（ms）に対する分布は、指数関数として次式で表される。

$$y(\Delta t) = \exp\left(p_0 + p_1, \Delta t_{\mathrm{ms}}\right), \qquad (1.5 \leq \Delta t_{\mathrm{ms}} \leq 10). \tag{6.1}$$

フィットにより得られたパラメータは以下の通りである。

$$p_0 = 16.6387 \pm 0.0009, \qquad p_1 = -\left(1.69486 \pm 0.00043\right)\ \mathrm{ms}^{-1}. \tag{6.2}$$

指数分布における率定数 λ（単位時間あたりの平均イベント数）は

$$\lambda = -p_1 \quad (\mathrm{ms}^{-1}) \tag{6.3}$$

と定義される。したがって、イベントレート R（Hz）は単位変換より

$$R = 10^3 \lambda = -10^3 p_1 \tag{6.4}$$

となる。数値を代入すると、

$$R = (1.69486 \pm 0.00043) \times 10^3 = (1.6949 \pm 0.0004) \times 10^3\ \mathrm{Hz} \tag{6.5}$$

が得られる。以上より、Si-DSD を用いた長時間動作試験において取得された全イベントに対するイベントレートは 1694.86 ± 0.43 Hz であることが確認された。

一方、取得した全イベント数は 32,526,639 events であり、試験時間 54,000 s で割ると、平均取得レートは約 602.36 counts/s となる。この取得レートと前述のイベントレートを比較すると、Si-DSD におけるデータ取得効率は約 34.5

ここで、非麻痺型デッドタイムモデルを仮定すると、観測レートと取得レートの関係は次式で表される。

$$R_{\mathrm{obs}} = \frac{R_{\mathrm{in}}}{1 + R_{\mathrm{in}}\tau} \tag{6.6}$$

ここで、R_{obs} は取得レート、R_{in} は真のイベントレート、τ はデッドタイムを表す。この式を τ について解くと、

$$\tau = \frac{1}{R_{\mathrm{obs}}} - \frac{1}{R_{\mathrm{in}}} \tag{6.7}$$

となる。$R_{\mathrm{obs}} = 602.36$ counts/s、$R_{\mathrm{in}} = 1694.86$ counts/s を代入すると、

$$\tau \simeq 1.07 \times 10^{-3}\ \mathrm{s} \tag{6.8}$$

すなわち、Si-DSD のデッドタイムは約 1.07 ms であることが分かる。この値は、待ち時間ヒストグラムの指数分布の傾きから推定されるデッドタイムと整合している。

なお、ここで求めたイベントレートは、6.4.2 節で示した Si-DSD を用いた長時間動作試験におけるライトカーブの平均カウントレート（約 308 counts/s）よりも大きい。これは、本ヒストグラムには物理イベントに加えて Pseudo イベントおよびノイズイベントが含まれているためである。

6.5 まとめ

本章では、Si-DSD を用いた長時間動作試験の結果について示した。CdTe-DSD と同じ ASIC を搭載した Si-DSD を用いて、^{241}Am 線源を用いて X 線を照射し、15 時間にわたり連続でデータを取得した。取得したデータからは、正常にスペクトルやイメージングを作成することができた。さらに、取得したデータの長期安定性を評価した結果、15 時間にわたりデータが欠損したり、システムが停止することなく、安定してデータを取得できることを確認した。2028 年に予定されている SuperHERO 気球実験では、約 2 日のフライト中に、最長半日程度の連続観測が予定されているため、Si-DSD を用いた 15 時間の長時間動作試験の結果は SuperHERO 気球実験成功に向けて十分な性能と安定性を有していることを示している。

第7章　結論

本研究は、SuperHERO 気球実験を成立させるために不可欠であった DE ソフトウェアを新規に設計・実装し、その実運用耐性を実証した研究である。

硬 X 線の高分解能の撮像観測は、SNR や PWN の加速現場付近での物理現象の解明に重要な役割を果たすと期待されている。SuperHERO は NuSTAR や Hitomi/HXI が可能とする空間分解能の約 10 倍の高い空間分解能を実現し、硬 X 線帯域での科学的なブレークスルーをもたらすことが期待されている。現在は 2028 年秋にアメリカでの気球実験を目指して開発が進められており、そこでは約 10 秒角の空間分解能を達成することを目標としている。しかし SuperHERO ミッションでは使用する CdTe-DSD 検出器と、使用する NiCo ミラーが決定されていたが、その検出器からデータを取得し CdTe-DSD を制御する DE ソフトウェアが未完成であった。

今回、 SuperHERO ミッションに向けて最後のピースとなる DE ソフトウェアの開発を一から行い、実際の検出器での 15 時間長時間試験を成功させた。15 時間の長時間試験では、データの欠損や異常動作を一切発生させることなく安定してデータを取得し続けることに成功した。2028 年秋の気球実験では、最長で半日程度の連続観測が見込まれており、15 時間の長時間運用の成功は気球実験成功に向けて十分な性能を有していることを示す結果となった。

さらに、DUA の性能評価と DE ソフトウェアの性能評価を通じて DE ソフトウェアが DUA の限界性能の 90% 以上を達成していることを確認した。2028 年秋の気球実験では Crab PWN を主な観測ターゲットとしており、その観測レートは約 2.7 counts/s と見積もられている。DE ソフトウェアはそれを大きく上回る約 29,000 counts/s の性能を有しており、約 1 万倍の余裕を持って観測を行うことが可能である。第二ターゲットである Hercules X-1 も Crab PWN と比べて低い観測レートが見込まれており、DE ソフトウェアは十分な性能を有している。

CdTe-DSD 内部のデッドタイムを考慮した場合においても、DE ソフトウェアの性能は十分であると結論付けることができる。CdTe-DSD の典型的なデッドタイムは約 330 µs であり、この値から見積もられる CdTe-DSD 一枚あたりの最大処理可能イベントレートは約 300 counts/s である。一方で、本研究で開発した DE ソフトウェアは約 29,000 counts/s の処理能力を有しており、仮にすべての検出器が限界レートで同時に動作する極端な条件を仮定した場合であっても、90 台程度の CdTe-DSD を同時に処理可能な性能余裕を有している。したがって、将来数十台規模の CdTe-DSD を搭載したミッションへ発展した場合においても、各検出器を物理的限界まで動作させた状況下で、DE ソフトウェアがデータ取得系のボトルネックとなることはなく、十分な処理性能と拡張性を備えていることが示された。

また、DE ソフトウェアの開発と並行して DUA シミュレータの作成も同時に行い、DE 単体の動作検証に留まらず、NASA 側を含むシステム全体の統合試験を可能にする基盤を構築した。これにより、NASA 側においても DEU の設計および動作テストを実機に依存せずに進めることが可能となり、DEU の開発を円滑に進める上で重要な役割を果たした。さらに、本シミュレータは運用試験や観測シナリオに応じた各種テストにも活用可能であり、SuperHERO ミッション全体の開発効率および信頼性向上に寄与する成果である。

以上の結果から、本研究で達成した主な成果を以下にまとめる。

(1) SuperHERO 気球実験を成立させるために不可欠であった CdTe-DSD 用 DE ソフトウェアを新規に設計・実装し (3 章)、複数台の CdTe-DSD (DUA) からのデータ収集・集約・転送を行うソフトウェアアーキテクチャを構築して実機環境で動作させた (6 章)。

(2) DEU の開発を目的として DUA シミュレータを新規に開発し、NASA 側における DEU の設計および動作テストを可能にすることで、システム全体の統合試験ならびに DEU 開発に実質的に貢献した (4 章)。

(3) 実際の Si-DSD を用いた 15 時間にわたる長時間連続動作試験および性能評価を行い、データ欠損や異常動作のない安定動作を実証するとともに、DUA の限界性能の 90% 以上を達成し、実観測要求に対して約 1 万倍の性能余裕を有することを定量的に示した (5 章)。

謝辞

令和 8 年 1 月 4 日藤本 源

本修士論文の執筆および研究の遂行にあたり、多くの方々から多大なるご指導とご支援を賜りました。ここに深く感謝の意を表します。

まず、本研究全体を通して終始丁寧かつ的確なご指導を賜りました指導教員の馬場彩准教授に心より御礼申し上げます。研究の方向性に迷った際には常に本質的な視点から助言を下さり、多くのことを学ばせていただきました。また、日常的な研究活動においてきめ細やかな助言と実務的な支援をして下さった助教の萩野浩一先生にも深く感謝いたします。実験に関する技術的な指導のみならず、議論や検討を重ねる中で得られた知見は、本研究の完成に不可欠なものでした。

本研究に関連する実験においては、渡辺伸准教授に多大なるご協力を賜りました。実験環境の提供のみならず、専門的見地からの貴重なご意見やご指摘により、研究の精度と信頼性を大きく高めることができました。ここに厚く御礼申し上げます。SuperHERO 計画に関する技術的な部分には渡辺先生自身が作成したプログラムも多く含まれており、その理解に関しても多くの助力をいただきました。

また、本研究の基盤となる SuperHERO 計画に参加する機会を与えて下さった高橋忠幸教授に深く感謝いたします。本計画への参加を通じて、国際的かつ実践的な研究開発の現場を経験できたことは、今後の研究者人生においても極めて貴重な財産となりました。

さらに、実験や日々の研究活動において協力して下さった研究室の皆様にも心より感謝いたします。議論や情報交換、作業面での助け合いを通じて、研究を前向きに進めることができました。研究室という恵まれた環境の中で本研究を遂行できたことを、ここに感謝をもって記します。

最後に、これまで支えて下さったすべての方々に改めて感謝申し上げるとともに、本論文をその成果としてここに提出いたします。

付 録A　SpaceWire のルーティング

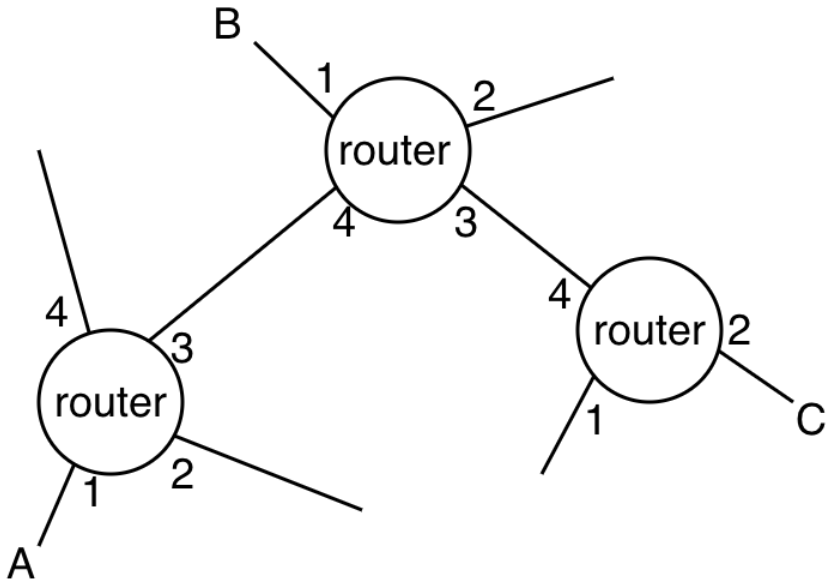


図 A.1: 各ノードと Router との関係図

SpaceWire では、ルーターを介して目的のノードへパケットを送信する。図 A.1 は、複数のノードが SpaceWire ルーターを介して相互に接続されている構成を示したものである。図中の「Router」と記された部分は、各ノードへの分岐機能を担う SpaceWire ルーターを表している。このノード群において、A、B、C はノードを示し、Router は SpaceWire ルーターを示す。以下では、A、B、C をノード、Router をルーターと呼ぶこととする。

A.1　Path Addressing

SpaceWire のルーティング方式の一つに Path Addressing がある。これは、送信元から宛先までに通過する各ルーターでの分岐方向を Path Address として列挙し、その列（Destination Address）をパケット先頭に格納して転送する方式である。Path Address は 1〜31 の整数で表され、各ルーターの出力ポート番号に対応する。パケットは各ルーターを通過するたびに、Destination Address の先頭要素が取り出され、その値に対応するポートへ転送される。

RMAP で Path Addressing を用いる場合、要求パケットの Destination Address を Target SpaceWire Address、応答パケットの Destination Address を Reply Address と

して指定する。例えば図 A.1 の A から B に送信する場合、Target SpaceWire Address は [3, 1]、Reply Address は [4, 1] である。同様に、A から C に送信する場合、Target SpaceWire Address は [3, 3, 2]、Reply Address は [4, 4, 1] となる。RMAP 仕様では Target SpaceWire Address に指定できる個数に制限はないが、Reply Address は最大 12 個までである。このため、返信経路に関してはルーティング深さの上限は 12 となる。

A.2 Logical Addressing

SpaceWire のもう一つのルーティング方式に Logical Addressing がある。Logical Address は、ネットワーク内の各ノードを識別するために割り当てられる論理アドレス（32〜255）である。各ルーターは、Logical Address と出力ポート（Path Address）の対応関係を示すルーティングテーブルを保持しており、受信したパケットの Logical Address を参照して、適切なポートへ転送する。これを参照することで目的のノードにパケットを送信することができる。

A.3 SuperHERO での Addressing

Path Addressing の利点は、ルーティングテーブルを必要とせず、各ルーターに対する事前設定を不要とできる点にある。このため、小規模なネットワークでは構成変更時の再設定作業が簡素化され、SpaceWire ネットワークの組替えが容易となる。一方で、ネットワーク規模の拡大に伴い、各ノードへの Path Address の組み合わせ数が増加し、管理の煩雑化を招く。その結果、一定規模を超えると、Path Addressing による個別管理よりも、ルーティングテーブルを用いて集中管理する Logical Addressing の方が運用上有利となる。

SuperHERO では、7 台の CdTe-DSD、2 台の SpaceWire Router、2 台の DE を用いる構成であり、合計 11 ノードからネットワークが構成される。この規模では、Path Addressing と Logical Addressing のいずれを採用しても管理の複雑さに本質的な差は小さいと判断される。したがって、本システムでは、設定不要で簡潔な運用が可能な Path Addressing を採用する。

ただし、各 CdTe-DSD を個別に識別するため、DE では各検出器にユニークな Logical Address を割り当て、ソフトウェア上の識別子として用いている。この設計により、DE においてルーティングテーブルを適切に設定すれば、Logical Addressing を用いた通信も可能である。将来的に検出器数の増加やネットワーク構成の高度化が生じた場合には、運用性および保守性の観点から、Logical Addressing の本格的な導入について検討する必要がある。

付 録B　読み出し系システムを開発するにあたっての設計思想

必要要件を満たす以上に、より発展的なシステムを構築するために、以下の設計思想を採用した。特に SpaceWrie RMAP Library [22] ではなく新規に SpaceWire 通信ライブラリを開発した背景には、これらの設計思想が深く関わっている。

まず第一に、ハードウェア単位だけではなくソフトウェアにおいてもモジュール化を念頭に置いた設計を行った。互いのコンポーネントが疎結合になるように設計し、通信部分と制御部分を明確に分離することで、将来的な拡張や保守を容易にした。例えば、SpaceWire の通信部分は今回一から設計をし、依存のない C++ のライブラリとして完成させた。

gRPC を使用した DE / DEU 間通信も同様の設計思想をもとに構成している。gRPC に限らず、 RPC とは遠隔で手続きを呼び出す仕組みであり。データ収集やコマンド処理の内部として通信に依存することなく設計することができる。

そして第二に、例外を使用しないエラーハンドリングである。C++ は強力な例外機構を持っているが、リアルタイム性が求められるシステムには不向きである上に、例外の補足漏れはシステムのプロセスを完全に停止させて、復帰不可能な状態に陥らせる可能性がある。特に、 依存するライブラリがスローする例外を補足できない場合、システム全体がクラッシュする可能性がある。これは、出来る限りの依存関係を排除し、システム全体の安定性を確保する設計思想とも関連しており、chapter D で述べる SpaceWire 通信ライブラリを一から設計した理由の一つでもある。

C++20 からは std::expected という、成功値と失敗値を同時に扱う型が標準ライブラリに追加された。C++23 からは monadic な操作も追加され、素早く呼び出し元にエラーを伝達することができるようになった。gRPC の呼び出し時も std::expected を用いてエラーハンドリングを行い、クライアントに適切にエラーを伝達できるようにした。こうすることで、クライアント側から正確にエラーの内容を把握でき、適切な対処を行うことができる。

将来的には一般的な Linux システムではなく、組み込みシステムやリアルタイム OS 上で動作させる可能性もある。その環境化では例外機構がサポートされていない可能性もあるため、例外を使用しない設計は将来的な移植性の向上にも寄与する。

そして第三に、 heap メモリの動的確保を最小限に抑える設計である。第二の例外を使用しないという設計思想と関連しているが、 heap メモリの動的確保は限られたリソースしか持たない組み込みシステムにおいては、パフォーマンスの低下やメモリ断片化を引き起こす可能性がある。ランタイムに予測不可能なメモリ確保は高負荷な処理が並行で走った場合、例外を引き起こす可能性があり、システムの安定性を損なう。そのため、可能な

限りスタックを使用し、 heap を使用する場合はプログラムの開始時に一括で確保し、動的な確保を避ける設計を行った。

そして第四に、ゼロコピー設計である。第三の heap メモリの動的確保を最小限に抑える設計と関連しているが、データのコピーは CPU 負荷を増大させ、システムのパフォーマンスを低下させる。特に heap メモリのコピーはメモリ使用量を増大させ、意図しないメモリリークを引き起こす可能性がある。TCP 通信や gRPC 通信に乗せる最小限のデータ以外はコピーを避け、可能な限り参照を用いる設計を行った。この設計は実行時の性能とメモリ使用量を削減する上に、システム全体として予測可能な動作を実現し、安定な動作を保証する。

付 録C　コンパイル安全性と実行時安全性

ソフトウェアの安全性を担保するにあたって、コンパイル時に安全を担保するコンパイル安全性と、実行時に安全を担保する実行時安全性の二つの概念がある。コンパイル安全性とは、プログラムがコンパイルされる段階で、型システムや静的解析ツールを用いて、プログラムの正当性を検証することである。例えば、 C++ の型システムは、変数の型が一致しない場合にコンパイルエラーを発生させることで、型の不一致によるバグを防止する。

C++ では強力な型システムを持っているが、コンパイル安全性を完全に担保することは難しい。特にポインタ操作やキャストなどの低レベルな操作を行う場合などが顕著になる。TCP 通信においては、受信したデータはバイト列として扱われるため、適切にデシリアライズしないと、型の不一致やバッファオーバーフローなどの問題が発生する可能性がある。特に、 Byte 列から構造体に変換する際に、データの整合性を検証しないと、不正なデータが構造体に変換され、プログラムの動作が不安定になる可能性がある。

gRPC が使用している Protocol Buffers は、このコンパイル安全性の問題をある程度解決している。Protocol Buffers は、 proto ファイルでデータ構造を定義し、その定義に基づいてコードを自動生成し、シリアライズとデシリアライズの処理の同一性を保証する。これはつまり、通信側で格納したデータが、受信側で正しく解釈されることを保証する。受信側は受信したデータを C++ の安定な「型」システムの上で扱うことができるため、この型のサイズや構造に関しての不整合が発生する可能性を大幅に低減できる。これが gRPC を採用することで得られる型安全性の利点である。

一方、実行時安全性とは、プログラムが実行される段階での安全性の担保であり、エラーハンドリングや例外処理を通じて、プログラムの正当性を検証することである。例えば、 C++ の例外機構は、プログラムが異常な状態に陥った場合に、例外をスローし、適切なエラーハンドリングを行うことで、プログラムの安定性を維持する。特に、安定したシステムにおいては、メモリーリーク、デッドロック、競合状態、範囲外アクセスによるクラッシュなどのソフトウェアを直接的に停止させる問題を防止するために、実行時安全性が重要となる。

C++ では、スマートポインタや標準ライブラリのコンテナクラスなどを活用することで、実行時安全性をある程度担保することができる。DE の開発でも `std::vector`、`std::unique_ptr`、`std::shared_ptr` などのスマートポインタを活用し、メモリ管理を自動化することで、メモリリークやダングリングポインタなどの問題を防止している。複数のスレッドが同時にアクセスするデータに対しては、ミューテックスや条件変数を用いて、競合状態やデッドロックを防止している。管理するスレッド数を意図的に制限し、複

雑な同期処理を避ける設計も有用であり、DE の設計においても可能な限りシンプルなスレッドモデルを採用している。

このコンパイルと実行時の安全性の両方を高いレベルで担保することが、信頼性の高いソフトウェアシステムを構築する上で重要である。Rust に始まる近年の言語設計では、所有権システムや借用チェッカーなどの機能を通じて、メモリ安全性をコンパイル時に担保する設計が採用されている。C++ では、スマートポインタやコンテナクラスなどの標準ライブラリを活用することで、メモリ安全性をある程度担保することができるが、その安全性を担保したまま設計を行うには高度な知識と経験が必要となる。未だ C++ が広く使用されている背景には、その高い性能と柔軟性があるが、将来的な移植性や保守性を考慮すると、より安全な言語への移行も検討する必要があり、その候補として Rust などを使用することも十分視野に入れる必要がある。

付 録D　spw_rmap の開発

D.1　背景と設計思想

DE の設計にあたり、SpaceWire RMAP 通信には、従来より広く利用され、宇宙機ミッションにおいても実績を有する SpaceWireRMAPLibrary [22] の存在を踏まえた上で新規実装を取り入れることとした。ここではその背景と設計思想について述べる。

SpaceWireRMAPLibrary は C++ により実装された成熟した SpaceWire RMAP 通信ライブラリであり、長年にわたり信頼性の高い通信機構を提供してきた実績を有する。特にプロトコル実装の正確性や汎用性の観点では高く評価されており、SpaceWire RMAP 通信の標準的な実装として広く採用されている。

一方で、本研究で対象とする DE では、複数検出器に対して高頻度かつ大容量の read/write 処理を長時間にわたり安定して継続する必要があり、通信処理における実行特性がシステム全体の性能に与える影響が大きいことが想定されていた。

特に 32KB 規模の RMAP 転送を高い周期で繰り返す運用では、通信ライブラリ内部におけるメモリコピー回数、動的メモリ確保、エラー処理の分岐が、レイテンシの揺らぎやスループット低下の要因となり得る。

さらに、SpaceWireRMAPLibrary が提供する SSDTP2 機能はは主としてテスト段階での利用を想定した実装であり、実際の観測機に組み込んで連続運用する用途に対しては、挙動の予測性や安定性の面で課題が残る。

そして SuperHERO の DE シミュレータの開発において、SpaceWireRMAPLibrary には存在しない SSDTP2 のサーバー側の実装が必要となり、SpaceWireRMAPLibrary に新たな機能を追加するよりも、一から専用の通信ライブラリを開発する方が効率的であると判断された。

このような背景から、本研究では SuperHERO において要求される高速かつ予測可能な実行特性に特化した SpaceWire RMAP 通信実装が必要であると判断した。

そこで本研究では、高速な SpaceWire RMAP 通信処理を実現することを主目的として、新たに SpaceWire RMAP 通信ライブラリ `spw_rmap` [20] を開発した。本ライブラリでは、以下の設計思想を採用している。

- **固定長バッファと逐次処理**送受信バッファをあらかじめ確保し、同一バッファ上で RMAP パケットの生成から解析までを完結させる構造とした。ユーザが結果を取得する段階でのみ必要最小限のコピーを行うことで、データ転送量の増加を抑え、安定した高スループットを実現している。

- **例外を用いないエラー処理** C++23 に準拠した `std::expected` を用い、成功値と

エラー値を明示的に返却する設計とした。これにより制御フローが明確となり、高速動作を維持したまま通信断などの異常状態からの復帰を可能としている。

- **依存関係の最小化**標準ライブラリおよび posix socket のみに依存する構成とすることで、余分な抽象化や処理を排除し、実行時オーバーヘッドを低減した。
- **明確な所有権管理**メモリ管理は `std::vector` や `std::unique_ptr` によって一元化し、API からはコピー済みデータまたは読み取り専用ビューのみを提供することで、安全性を維持しつつ高速な処理を可能とした。

このように `spw_rmap` は、既存の信頼性ある SpaceWireRMAPLibrary 実装の知見を踏まえつつ、SuperHERO において特に重要となる通信性能と実行特性の安定性を重視して設計されたライブラリである。DE における通信性能を最大限に引き出すことを目的とし、結果として高いスループットと予測可能なレイテンシを実現している。

D.2　spw_rmap の使用例 C++

以下には spw_rmap を用いて RMAP 読み出しを行うサンプルコードを示す。

図 D.1: spw_rmap を用いた RMAP 読み出しの例 (C++)

```cpp
#include <array>
#include <chrono>
#include <iostream>

#include "spw_rmap/spw_rmap_tcp_node.hh"
#include "spw_rmap/target_node.hh"

auto main() -> int {
  spw_rmap::SpwRmapTCPClient client(
      {.ip_address = "192.168.1.100", .port = "10030"});
  // 自動ポーリングモードを有効化
  client.setAutoPollingMode(true);

  // 接続
  auto res = client.connect(std::chrono::milliseconds(500));
  if (!res) exit(1);

  // ターゲットノードの作成
  auto target = spw_rmap::TargetNode(0x32)
    .setTargetAddress(0x06, 0x02)
    .setReplyAddress(0x01, 0x03);

  // 書き込み
  res = client.write(target, 0x44A20000, {0x01, 0x02, 0x03, 0x04});
  if (!res) exit(1);

  // 読み込み
  std::array<uint8_t, 4> buf{};
  res = client.read(target, 0x44A20000, buf);
  if (!res) exit(1);

  for (auto b : buf) std::cout << " 0x" << std::hex << +b;

  // シャットダウン
  res = client.shutdown();
  if (!res) exit(1);
  return 0;
}
```

D.3 spw_rmap の使用例 Python

以下には spw_rmap を用いて RMAP 読み出しを行う Python サンプルコードを示す。spw_rmap は Python バインディングも提供しており、 Python からも同様の機能を利用できる。この python バインディングは pyspw_rmap [23] として PyPI に公開されている。pip を用いて pip install pyspw_rmap とすることで簡単にインストールでき、Python 環境から RMAP 通信を行うことができる。

図 D.2: spw_rmap を用いた RMAP 読み出しの例 (Python)

```python
from pyspw_rmap import TargetNode, SpwRmapTCPNode

target = TargetNode(
    logical_address=0x32,
    target_spacewire_address=[0x06, 0x02],
    reply_address=[0x01, 0x03],
)

spw = SpwRmapTCPNode(ip_address="192.168.1.100", port="10030")

try:
    spw.write(target, 0x44A200D4, [0, 0, 0, 1])
    print(spw.read(target, 0x44A200D0, 4))
except Exception as e:
    print(f"An error occurred: {e}")
```

付 録E　Nix を使用した DE

Nix は関数型パッケージマネージャーであり、 OS やライブラリ、アプリケーションなどのソフトウェアを宣言的に管理することができるツールである。一般的にはパッケージマネージャーとして使用されることが多いが、Nix 自体は宣言的で再現性の高いシステム構築を可能にするものであり、 OS 全体を Nix で記述することも可能である。

ビルドプロセスから、起動プロセス、そして Raspberry Pi の細かい設定自体も全て Nix で記述し、コマンド一つで DE のシステム全体を再現できるようにしている。

DE を開発し運用していく上でこれには大きく二つの利点がある。

まず、第一に NASA の開発者との共有が容易になることである。NASA の開発では、DE 及び DUA のシミュレータを用いた DEU の開発が行われている。それと同時に、日本側でも DE の開発や仕様の調整が行われる。その際にビルド環境や OS の設定が異なると、同じソースコードであっても動作が異なったり、コンパイルが通らなかったりする可能性がある。しかし、 Nix では再現性の高いビルド環境を提供できるため、 NASA 側と日本側で全く同じシステムを構築することが可能になる。

そして、第二に将来的に DE を複数台運用する際に、同じシステムを複数台に展開するのが容易になることである。Nix では OS の設定自体も宣言的に記述できるため、同じ Nix ファイルを用いて複数台の Raspberry Pi に同じシステムを展開することが可能になる。現在 Nix を用いて直接 Raspberry Pi の SD Card イメージを構築するようにしている。

DE の設計方針としてシンメトリックなシステムを目指しており、台数に限らず並列で全く同じシステムを動作させることができるようにしている。将来的に DE を複数台運用する際に、 作成した SD Card イメージを複数台に展開するだけでより広い検出器ネットワークを構築できるようになる。

付 録F　DE を使用した地上試験用プログラムの開発

DEU は NASA 側で開発されており、 DE の動作を確認するためのプログラムは DE / DEU とは独立に開発する必要がある。DE の動作確認としての要素だけではなく、ビームライン試験時の簡単な UI と包括的な操作性を持つプログラムが必要である。CdTe-DSD には従来から存在していたコマンドラインベースの制御プログラムがあるが、一台の検出器に対してのみ動作し、複数台を同時に制御することはできない。

このインタラクティブなコマンドラインプログラム以下の 3 つ要求を見たすように設計した。

要求 1 (完全性) DE の High Level / Low Level / DataStream コマンドを全てサポートし、CdTe-DSD の全ての機能を操作できるようにした

要求 2 (操作性) 簡潔な UI と簡潔な動作で、始めてソフトを触る人でも直感的に操作できるようにした。

要求 3 (Batch ファイル) Batch ファイルをサポートし、複数のコマンドを一括で実行できるようにした。これは条件を変えながら複数の試験を行う際に有用である。

要求 1 の完全性に関しては、バグが存在するか確認するために全てのコマンドをサポートする必要がある。基本的にこのプログラムも High Level コマンドのみで観測ルーチンを実行することが可能である。一方で Low Lovel コマンドを用いて FPGA の設定を全て表示するなどの、詳細な動作確認も可能にしている。

要求 2 の操作性に関しては、では C++ の readline ライブラリを用いてインタラクティブなコマンドライン UI を提供している。readline ライブラリを用いることで、コマンド履歴の保存やタブ補完などの機能を簡単に実装できる。特にタブ補完は登録した CdTe-DSD や Router の一覧から補完候補を表示するため、複数台の CdTe-DSD を操作する際に有用である。動作に関しては、観測したデータを上書きしないように自動でファイル名を変更する、観測時間、露光時間等の観測の詳細をファイルの拡張属性として保存し、後から容易に確認できるようにするなどの工夫を行っている。

要求 3 の Batch ファイルのサポートに関しては、コマンドライン引数で Batch ファイルを指定することで、そのファイル内に記述されたコマンドを順次実行することができるようにしている。Batch ファイル内ではコメント行や空行をサポートし、また他の Batch ファイルを呼び出すことも可能である。

図 F.1: DE を使用した地上試験用プログラムの Batch ファイル例

```
# DE と接続
connect 192.168.2.14:50051

# 既存のデバイスをすべて削除
remove_all_devices

# ルーターと検出器の追加
add_router 0x21 2 - 3
add_router 0x22 1 2 - 6 3
add_detector 0x34 1 1 2 - 1 6 3
add_detector 0x35 6 2 - 1 3
add_detector 0x36 7 2 - 1 3
add_detector 0x37 9 2 - 1 3
add_detector 0x38 10 2 - 1 3

# Low Level コマンドでのレジスタ設定
# (ここでは Pseudo レジスタを使用)
set PseudoONOFF  [0x35, 0x36, 0x37, 0x38] 0
set PseudoRate   [0x35, 0x36, 0x37, 0x38] 211
set PseudoONOFF  [0x35, 0x36, 0x37, 0x38] 1

# chip1.dat chip2.dat chip3.dat chip4.dat からレジスタ設定を読み込み
# ASIC のレジスターの設定をする
set_vareg 0x34 ../regfile/chip{}.dat

# FPGA の設定
configure_fpga 0x34 \
 peaking_time_nside=0x00E1 \
 peaking_time_pside=0x00FA \
 adc_clock_period=0x00000004 \
 readout_clock_period=0x0000000F \
 readout_clock_delay=0x00000016 \
 trip_patlatch_timing=0x0000000A \
 reset_wait_time=0x000001F4 \
 reset_wait_time2=0x000038A4

# 30 分間データ取得
readout 30m ./data/30min_data_aquisition
```

付 録G　取得データの解析プログラムの開発

DE から取得したデータはバイナリ形式で保存されている。イベントフレーム毎に 32,768 バイトのデータが連続して保存されているが、そのままでは人間が解析するのは困難である。生のデータを解析し、人間が理解しやすい形式に変換するために、ROOT 形式のファイルに変換するプログラムを開発した。このプログラムとしてもともと CdTe-DSD 用に開発された `spmu001_minisgd_vataread_v5_dram` が存在していたが、DE を開発するにあたってバイナリのフォーマットを変更したため、今回新たに DE 用のデータ解析プログラムを開発した。

イベント解析のプログラムの要件としては以下の 3 つがある。まず一つは、フレームを正しく解析を行い、各イベントの情報を正確に抽出できることである。DE から取得したバイナリデータは、各イベントフレームが連続して保存されている形式であり、各フレームのヘッダー情報やイベントデータを正確に解析し、各イベントの ADC 値や CMN 値、などの情報を抽出する必要がある。二つ目は、解析したデータを ROOT 形式で保存することである。ROOT 形式は高エネルギー物理学や宇宙物理学の分野で広く使用されており、データの可視化や解析に便利なツールが豊富に存在する。特に解析したデータをヒストグラムやグラフとして可視化することで、簡単にデータの傾向を把握できるようになる。イベントデータは ROOT の TTree 形式で保存し、各イベントの情報を効率的に格納できるようにしている。また Histgram として channel を横軸に ADC 値を縦軸に持つ 2D ヒストグラムと、channel を横軸に adc − cmn 値を縦軸に持つ 2D ヒストグラムも保存するようにしている。三つ目は、高速に動作することである。大量のデータを解析する際に、解析時間が長くなると効率的な解析が困難になる。例えば 6 で示した長時間動作試験では、15 時間のデータ収集で約 12 GB のデータが取得された。このデータを解析するのに 15 分以上時間がかかってしまうと、解析の効率が悪くなってしまう。

これらの 3 つの要件を満たした解析プログラム `raw2root` を新たに開発した。このプログラムは C++ で実装されており、ROOT ライブラリのみを依存関係として持っている。従来の `spmu001_minisgd_vataread_v5_dram` に比べて 3 倍以上高速に動作するプログラムであり、15 時間分の 12 GB のデータを約 5 分で解析できる性能を持っている。また、これを並列化して複数の CPU コアで同時に解析を行うことで、さらに高速に解析を行うことも可能であり、14 並列で動作させることで約 35 秒で 15 時間分のデータを解析できる。一時間程度のデータであれば、数秒で解析が完了するため、データを取得しデータを解析し、次の実験を行うサイクルを高速に回すことができる。

付 録H　エネルギー較正

取得したデータからスペクトルやイメージングを作成するためには、出力された ADC 値をエネルギーに変換するエネルギー較正が必要である。表 H.1 に ^{241}Am 線源照射時に観測される ^{237}Np 起源の輝線を示す。これらの輝線のうちピークとして観測される、13.81 keV, 17.8 keV, 20.8 keV, 26.34 keV, 59.54 keV を用いてエネルギー較正を行った。

表 H.1: ^{241}Am 線源照射時に観測される ^{237}Np 起源の輝線 ([4] より抜粋)

起源核種	線種・系列	エネルギー [keV]
^{237}Np	核 γ 線 $\gamma_{10,9}$	13.81(2)
^{237}Np	核 γ 線 $\gamma_{2,1}$	26.34460(24)
^{237}Np	L 特性 X 線 Lℓ	11.89
^{237}Np	L 特性 X 線 Lα	13.76–13.944
^{237}Np	L 特性 X 線 Lη	15.876
^{237}Np	L 特性 X 線 Lβ	16.13–17.79
^{237}Np	L 特性 X 線 Lγ	20.12–22.2

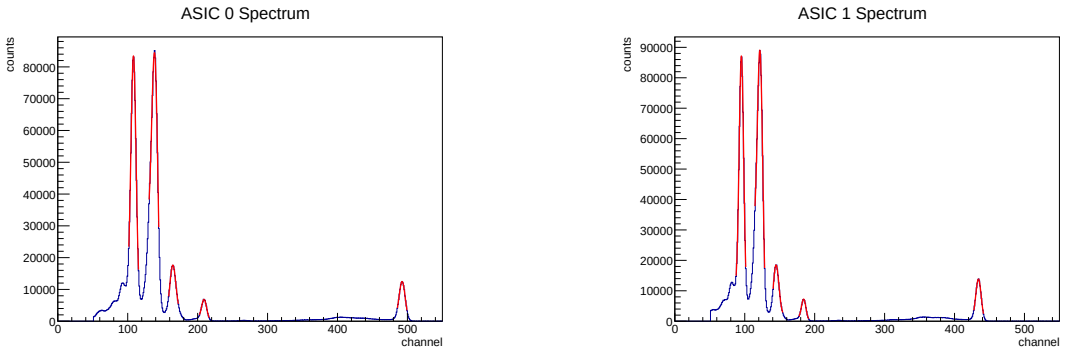


図 H.1: ASIC0 (左) と ASIC1 (右) 側のスペクトルとピークのフィッティング結果

図 H.1 に Si-DSD の ASIC0 側と ASIC1 側のスペクトルを示す。ASIC0 と ASIC1 は Si-DSD の ground 側の channel からのデータを収集している。本来であらば、各 ASIC に存在する channel ごとにエネルギー較正を行い、計 256 チャンネルで個別でのエネルギー

較正を行う必要がある。本試験ではしおりを用いてマスクをかけているため、各 channel ごとに統計数にばらつきが大きい。また、イメージング目的ではそこまで高いエネルギー分解能は必要ないため、ASIC ごとにまとめてエネルギー較正を行った。

図 H.1 に示す。赤のガウシアン曲線は各ピークに対してフィッティングを行った結果である。フィッティングでは gaussian + linear を用いている。^{241}Am の低エネルギー側の 4 つのピークはそれぞれ近くに存在しているため、ピーク同士の影響を考慮してなるべくピーク付近のデータのみを用いてフィッティングを行っている。

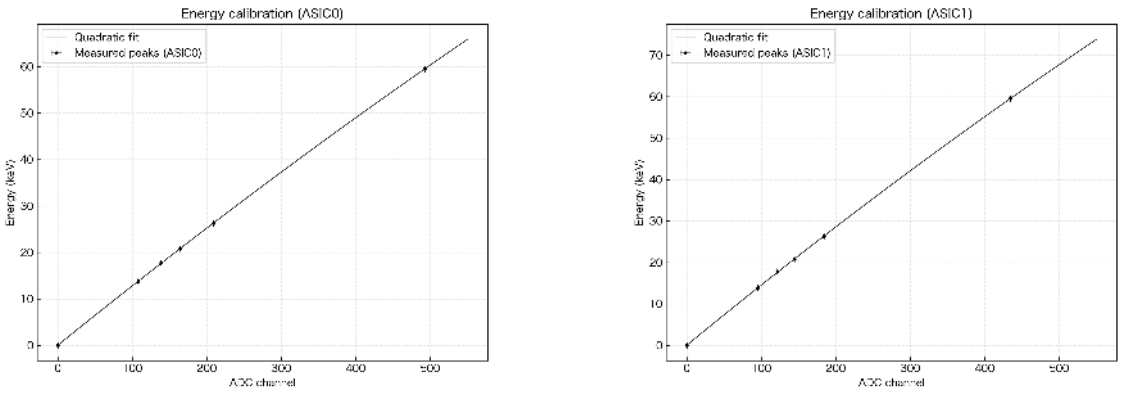


図 H.2: ASIC0 (左) と ASIC1 (右) 側のエネルギー較正曲線

図 H.2 に ASIC0 側と ASIC1 側のエネルギー較正を示す。横軸が ADC 値、縦軸がエネルギー (keV) である。各ピークのフィッティング結果から得られたピーク位置を用いてエネルギー較正を行った。今回は、得られたデータ点が少ないことと、それが 10–50 keV の範囲に集中していることから、簡単のためエネルギー較正は原点を通る二次関数でフィッティングを行っている。

付 録I　開発にあたって初期設計から修正を要した課題

DE の開発過程では設計時に想定していた方針が、実際に使用してみると必ずしも最適でないことが判明することが多々あった。ここではその中でも設計に大きく影響を与えたものについて述べる。

I.1　High Level コマンドの設計

開発初期段階では、通信インタフェースは HLSubmit、HLResult、LLSubmit、LLResult の四種類のコマンドのみで構成されており、設定値や制御情報は HLSubmit 等に生のバイト列として直接格納する方式を採用していた。この段階では機能数が限定的であったため、設計は比較的単純であり、動作確認や試験も容易であった。しかし、開発の進展に伴い制御機能や取得情報が増加すると、各コマンドに渡す引数は次第に複雑化し、バイト列の解釈や管理に多くの注意を要するようになった。特に High Level コマンドでは、処理結果を得るために HLResult を明示的に呼び出す必要があり、ソフトウェア実装時にはコマンド送信と結果取得の対応関係を常に意識しなければならなかった。その結果、ユーザコードは冗長になり、制御フローも分かりにくくなるという問題が顕在化した。

HLSubmit と HLResult は内部で TaskQueue に post する処理と、TaskQueue から結果を取得する処理に対応しており、非ブロッキングな設計となっていた。しかしこの非ブロッキングな設計では、結果取得のタイミング管理を実装側で行う必要があり、処理順序の誤りや取得漏れといったバグが発生しやすい構造であった。特にクライアント側の処理としてはボイラープロート的に何回も HLResult を呼び出すコードが散見され、可読性や保守性の低下を招いていた。これらは特に長時間運用や複雑な制御シーケンスにおいて、ソフトウェアの信頼性低下につながる懸念があった。

以上の背景から、本研究では全てのコマンドを独立したリモートコールとして再設計し、High Level コマンドについてはブロッキング動作を基本とする方式を採用した。この変更により、単一の呼び出しで処理の実行と結果取得が完結し、制御フローが直線的かつ直感的に記述可能となった。High Level コマンドはモード遷移や設定変更などの、アトミックであるべき操作が多いため、ブロッキング動作は特に適していると判断された。

I.2 TimeCode とスケジューリング

開発初期段階では、TimeCode に同期して同一スレッド内でデータ取得を実行する設計を検討していた。この方式は、TimeCode を基準としてデータ取得のタイミングを揃えられる点で直感的である一方、TimeCode 周波数が 64 Hz である条件下において、1 周期の間に何フレームの取得処理を確実に完了できるかを事前に見積もることが非自明であった。特に、通信遅延や処理時間のばらつきを考慮して安全マージンを確保すると、周期内の相当部分を TimeCode の待ち合わせに割り当てる必要が生じる。結果として、最悪ケースを想定した場合には、周期の半分以上が待ち時間となり、データ取得効率（スループット）を大きく損なう可能性があることが判明した。

TimeCode はネットワーク上で周期的に送信されるべきパケットであり、データ取得処理と強く同期させる必然性は低い。加えて、送信処理において send() に対する排他制御が適切に行われていれば、他の通信処理と並行して送信してもパケットの整合性や順序性に関する問題は生じない。

以上を踏まえ、本システムでは TimeCode 送信をデータ取得処理から分離し、別スレッドで TimeCode を個別に送信する設計を採用した。この変更により、TimeCode 待ち合わせがデータ取得処理を直接ブロックすることを回避し、処理系全体としての並列性とスループットを向上させた。

I.3 Raspberry Pi 4 と PC との接続方法と IP アドレス管理

開発初期の段階では、PC と Raspberry Pi 4 間の通信において、Ethernet デバイスに固定 IP アドレスを設定する方式を採用していた。しかし、システムを組み直すたびに通信が成立したり成立しなかったりする不安定な挙動が観測され、安定した開発および試験の妨げとなった。

Raspberry Pi 4 には標準で Ethernet ポートが一つしか搭載されていないため、複数系統のネットワーク接続が必要な場面では USB to Ethernet アダプタを併用する構成を採用した。しかし、この USB 接続では、使用する USB レセプタクルの物理的な位置や接続順序によってネットワークインタフェース名や設定が変化し、固定 IP 設定が意図通り適用されないケースが発生した。この結果、同一の設定ファイルを用いていても、接続可否が環境依存で変動する問題が顕在化した。

この問題に対する根本的な解決策として、本システムでは mDNS を使用することにるる IP アドレスの固定に依存しない通信方式を採用した。具体的には、 による名前解決を用い、ホスト名を基準として接続先を指定する設計へ移行した。これにより、ネットワークインタフェース構成や IP アドレスの割り当てが変化した場合でも、安定して Raspberry Pi 4 を認識・接続できるようになった。

I.4　高負荷時における Read データ破損問題

初期の負荷試験において、FPGA のリングバッファが溢れる水準までデータを生成した場合、読み出されたデータの内容が途中から別のデータに置き換わる現象が確認された。この問題は、一定時間までは正しいデータが取得できるものの、負荷が増大すると突然データの連続性が失われるという形で顕在化した。これは FPGA 側の実装が、DRAM への書き込み処理に対する排他制御（ロック）が行われていないことが原因であった。このため、FPGA が DRAM にデータを書き込み中の領域、特に write_pointer 近傍のデータに対して、読み出し側が同時にアクセスしてしまう状況が発生し、依然のデータが中途半端に上書きされる形でデータ破損が生じていた。リングバッファ構造においては、書き込み中の領域を読み出すことは未定義動作となり、結果として読み出しデータの破損を引き起こす。

FPGA 実装の制約上、DRAM 書き込みに対して厳密なロック機構を導入することは困難であったため、本研究では読み出し側の動作を制御する方針を採用した。具体的には、write_pointer と read_pointer の差が一定値 以下にまで縮まった場合には、データ取得を継続せず、意図的に読み出し量を切り詰める処理を追加した。本 DE では 20% のマージンを設けており、64 MB のうち、約 12.8 MB をアクセス不可領域として予約している。これにより、読み出し処理が書き込み中の領域に踏み込むことを回避している。

この設計変更により、高負荷条件下においてもデータ破損は発生しなくなり、リングバッファを用いたデータ取得における安全性と予測可能性が大幅に向上した。

参考文献

[1] T. K. Gaisser. The Cosmic-ray Spectrum: from the knee to the ankle. In *Journal of Physics Conference Series*, volume 47 of *Journal of Physics Conference Series*, pages 15–20. IOP, October 2006.

[2] Nicholas E. Thomas, Hugo Allaire, Jeff A. Apple, Adrianah E. Appleman, Dhamar Astilla Munoz, Aya Bamba, Stephen D. Bongiorno, McKinley Brumback, Patrick R. Champey, Chien-Ting Chen, Renata S. Cumbee, Steven R. Ehlert, Gen Fujiomoto, Jessica A. Gaskin, Danielle N. Gurgew, Anthony R. Guillory, Kouichi Hagino, Philip Kaaret, Miho Katsuragawa, Fabian Kislat, Jeffery Kolodziejczak, Henric Krawczynski, Ioannis Liodakis, W. Peter Maksym III, Kaya Mori, Brian D. Ramsey, Panini S. Singam, Patrick Slane, Chet O. Speegle, Douglas A. Swartz, Tadayuki Takahashi, Shin' ichiro Takeda, Allyn F. Tennant, and Shin Watanabe. The SuperHERO (Super–High Energy Replicated Optics) high-angular resolution, hard-x-ray telescope. In Jessica A. Gaskin and Daniele Spiga, editors, *Optics for EUV, X-Ray, and Gamma-Ray Astronomy XII*, volume 13626, page 136260Q. International Society for Optics and Photonics, SPIE, 2025.

[3] Matt Newville, Nathan Whittington, Matteo Levantino, Christian Schlepuetz, Damian Guenzing, Max Rakitin, Sang-Woo Kim, et al. XrayDB: X-ray reference data in sqlite with python interface, 2025.

[4] LNE-LNHB/CEA. Recommended Nuclear Decay Data: Americium-241 (Am-241) Tables. Technical report, Laboratoire National Henri Becquerel (LNE-LNHB), CEA, 2010. Accessed 2026-01-03.

[5] Renyue Cen and Jeremiah P. Ostriker. Where Are the Baryons? ApJ, 514(1):1–6, March 1999.

[6] M. C. Weisskopf, B. Brinkman, C. Canizares, G. Garmire, S. Murray, and L. P. Van Speybroeck. An Overview of the Performance and Scientific Results from the Chandra X-Ray Observatory. PASP, 114(791):1–24, January 2002.

[7] K. Koyama, R. Petre, E. V. Gotthelf, U. Hwang, M. Matsuura, M. Ozaki, and S. S. Holt. Evidence for shock acceleration of high-energy electrons in the supernova remnant SN1006. Nature, 378(6554):255–258, November 1995.

[8] R. D. Blandford and J. P. Ostriker. Particle acceleration by astrophysical shocks. ApJ, 221:L29–L32, April 1978.

[9] Toshiki Sato and John P. Hughes. Freely Expanding Knots of X-Ray-emitting Ejecta in Kepler's Supernova Remnant. ApJ, 845(2):167, August 2017.

[10] A. Hacar, M. Tafalla, J. Forbrich, J. Alves, S. Meingast, J. Grossschedl, and P. S. Teixeira. An ALMA study of the Orion Integral Filament. I. Evidence for narrow fibers in a massive cloud. A&A, 610:A77, March 2018.

[11] Martin C. Weisskopf, J. Jeff Hester, Allyn F. Tennant, Ronald F. Elsner, Norbert S. Schulz, Herman L. Marshall, Margarita Karovska, Joy S. Nichols, Douglas A. Swartz, Jeffery J. Kolodziejczak, and Stephen L. O'Dell. Discovery of Spatial and Spectral Structure in the X-Ray Emission from the Crab Nebula. ApJ, 536(2):L81–L84, June 2000.

[12] Kristin K. Madsen, Stephen Reynolds, Fiona Harrison, Hongjun An, Steven Boggs, Finn E. Christensen, William W. Craig, Chris L. Fryer, Brian W. Grefenstette, Charles J. Hailey, Craig Markwardt, Melania Nynka, Daniel Stern, Andreas Zoglauer, and William Zhang. Broadband X-ray Imaging and Spectroscopy of the Crab Nebula and Pulsar with NuSTAR. ApJ, 801(1):66, March 2015.

[13] T. Takahashi and S. Watanabe. Recent progress in CdTe and CdZnTe detectors. *IEEE Transactions on Nuclear Science*, 48(4):950–959, August 2001.

[14] Shin-nosuke Ishikawa, Shin Watanabe, Taro Fukuyama, Goro Sato, Motohide Kokubun, Hirokazu Odaka, Shinya Saito, Tadayuki Takahashi, Kazuhiro Nakazawa, and Takaaki Tanaka. Development of Double-Sided CdTe Strip Detectors for γ-Ray Imaging and Spectroscopy. *Japanese Journal of Applied Physics*, 49(11R):116702, November 2010.

[15] Shin Watanabe, Shin-Nosuke Ishikawa, Hiroyuki Aono, Shin'ichiro Takeda, Hirokazu Odaka, Motohide Kokubun, Tadayuki Takahashi, Kazuhiro Nakazawa, Hiroyasu Tajima, Mitsunobu Onishi, and Yoshikatsu Kuroda. High Energy Resolution Hard X-Ray and Gamma-Ray Imagers Using CdTe Diode Devices. *IEEE Transactions on Nuclear Science*, 56(3):777–782, June 2009.

[16] H. Tajima, T. Nakamoto, T. Tanaka, S. Uno, T. Mitani, E. Docoutoesilva, Y. Fukazawa, T. Kamae, G. Madejski, D. Marlow, K. Nakazawa, M. Nomachi, Y. Okada, and T. Takahashi. Performance of a Low Noise Front-End ASIC for Si/CdTe Detectors in Compton Gamma-Ray Telescope. *IEEE Transactions on Nuclear Science*, 51(3):842–847, June 2004.

[17] European Space Agency. Spacewire – links, nodes, routers and networks. Technical Report ECSS-E-ST-50-12C, ESA, 2019.

[18] Shimafuji Electric Inc. Spacewire-to-gigabitetherr3. http://www.shimafuji.co.jp/products/spacewire/3065, 2026. Accessed 2026-01-03.

[19] Tadayuki Takahashi, Motohide Kokubun, Kazuhisa Mitsuda, Richard L. Kelley, Takaya Ohashi, Felix Aharonian, Hiroki Akamatsu, Fumie Akimoto, Steven W. Allen, Naohisa Anabuki, Lorella Angelini, Keith Arnaud, Makoto Asai, Marc Audard, Hisamitsu Awaki, Magnus Axelsson, Philipp Azzarello, Chris Baluta, Aya Bamba, Nobutaka Bando, Marshall W. Bautz, Thomas Bialas, Roger Blandford, Kevin Boyce, Laura W. Brenneman, Gregory V. Brown, Esra Bulbul, Edward M. Cackett, Edgar Canavan, Maria Chernyakova, Meng P. Chiao, Paolo S. Coppi, Elisa Costantini, Steve O'Dell, Michael DiPirro, Chris Done, Tadayasu Dotani, John Doty, Ken Ebisawa, Megan E. Eckart, Teruaki Enoto, Yuichiro Ezoe, Andrew C. Fabian, Carlo Ferrigno, Adam R. Foster, Ryuichi Fujimoto, Yasushi Fukazawa, Stefan Funk, Akihiro Furuzawa, Massimiliano Galeazzi, Luigi C. Gallo, Poshak Gandhi, Kirk Gilmore, Margherita Giustini, Andrea Goldwurm, Liyi Gu, Matteo Guainazzi, Daniel Haas, Yoshito Haba, Kouichi Hagino, Kenji Hamaguchi, Ilana M. Harrus, Isamu Hatsukade, Takayuki Hayashi, Katsuhiro Hayashi, Kiyoshi Hayashida, Jan-Willem den Herder, Junko S. Hiraga, Kazuyuki Hirose, Ann Hornschemeier, Akio Hoshino, John P. Hughes, Yuto Ichinohe, Ryo Iizuka, Hajime Inoue, Yoshiyuki Inoue, Kazunori Ishibashi, Manabu Ishida, Kumi Ishikawa, Kosei Ishimura, Yoshitaka Ishisaki, Masayuki Itoh, Masachika Iwai, Naoko Iwata, Naoko Iyomoto, Chris Jewell, Jelle Kaastra, Tim Kallman, Tsuneyoshi Kamae, Erin Kara, Jun Kataoka, Satoru Katsuda, Junichiro Katsuta, Madoka Kawaharada, Nobuyuki Kawai, Taro Kawano, Shigeo Kawasaki, Dmitry Khangulyan, Caroline A. Kilbourne, Mark Kimball, Ashley King, Takao Kitaguchi, Shunji Kitamoto, Tetsu Kitayama, Takayoshi Kohmura, Saori Konami, Tatsuro Kosaka, Alex Koujelev, Katsuji Koyama, Shu Koyama, Peter Kretschmar, Hans A. Krimm, Aya Kubota, Hideyo Kunieda, Philippe Laurent, Shiu-Hang Lee, Maurice A. Leutenegger, Olivier Limousin, Michael Loewenstein, Knox S. Long, David Lumb, Greg Madejski, Yoshitomo Maeda, Daniel Maier, Kazuo Makishima, Maxim Markevitch, Candace Masters, Hironori Matsumoto, Kyoko Matsushita, Dan McCammon, Daniel Mcguinness, Brian R. McNamara, Missagh Mehdipour, Joseph Miko, Eric D. Miller, Jon M. Miller, Shin Mineshige, Kenji Minesugi, Ikuyuki Mitsuishi, Takuya Miyazawa, Tsunefumi Mizuno, Hideyuki Mori, Koji Mori, Franco Moroso, Harvey Moseley, Theodore Muench, Koji Mukai, Hiroshi Murakami, Toshio Murakami, Richard F. Mushotzky, Housei Nagano, Ryo Nagino, Takao Nakagawa, Hiroshi Nakajima, Takeshi Nakamori, Toshio Nakano, Shinya Nakashima, Kazuhiro Nakazawa, Yoshiharu Namba, Chikara Natsukari, Yusuke Nishioka, Kumiko K. Nobukawa, Masayoshi Nobukawa, Hirofumi Noda, Masaharu Nomachi, Hirokazu Odaka, Hiroyuki Ogawa, Mina Ogawa, Keiji Ogi, Masanori Ohno, Masayuki Ohta, Takashi Okajima, Atsushi Okamoto, Tsuyoshi Okazaki, Naomi Ota, Masanobu Ozaki, Frits Paerels, Stéphane Paltani, Arvind Parmar, Robert Petre,

Ciro Pinto, Jelle de Plaa, Martin Pohl, James Pontius, Frederick S. Porter, Katja Pottschmidt, Brian Ramsey, Christopher Reynolds, Helen Russell, Samar Safi-Harb, Shinya Saito, Kazuhiro Sakai, Shin-ichiro Sakai, Hiroaki Sameshima, Toru Sasaki, Goro Sato, Kosuke Sato, Rie Sato, Yoichi Sato, and Makoto Sawada. Hitomi (ASTRO-H) X-ray Astronomy Satellite. *Journal of Astronomical Telescopes, Instruments, and Systems*, 4:021402, April 2018.

[20] Gen Fujimoto. spw_rmap: A spacewire/rmap helper library, 2025.

[21] Shin'Ichiro Takeda, Shin Watanabe, Takaaki Tanaka, Kazuhiro Nakazawa, Tadayuki Takahashi, Yasushi Fukazawa, Hajimu Yasuda, Hiroyasu Tajima, Yoshikatsu Kuroda, Mitsunobu Onishi, and Kei Genba. Development of double-sided silicon strip detectors (DSSD) for a Compton telescope. *Nuclear Instruments and Methods in Physics Research A*, 579(2):859–865, September 2007.

[22] Takayuki Yuasa. Spacewirermaplibrary.

[23] Gen Fujimoto. pyspw_rmap: Python bindings for spw_rmap, 2025. PyPI package, MIT License.