

修士論文

CMB 観測実験 GroundBIRD の焦点面検出器アライメントと
長期運用に向けた角度データ取得システムの最適化

京都大学 理学研究科 物理学・宇宙物理学専攻
物理学第二教室 高エネルギー物理学研究室

片岡 敬涼

2024年1月24日



概要

gaiyou

目 次

第 1 章 序論	1
1.1 CMB の異方性と現代宇宙論	1
1.1.1 CMB の温度異方性	1
1.1.2 Λ -CDM モデル	1
1.1.3 地平線問題	1
1.2 CMB の偏光とインフレーション理論	1
1.2.1 インフレーション理論	1
1.2.2 CMB の偏光モード	1
1.2.3 偏光 B モードの探索状況	1
1.3 CMB の偏光とニュートリノ質量和	1
1.3.1 光学的厚み τ	1
1.3.2 ニュートリノ質量和との縮退	1
1.3.3 偏光 E モードと τ	1
第 2 章 GroundBIRD 実験	2
2.1 実験概要	2
2.1.1 GroundBIRD 望遠鏡	2
2.1.2 独創的なスキャン戦略	2
2.1.3 物理ターゲット	2
2.2 現在の観測状況	2
2.2.1 検出器のフルアレイインストール	2
2.2.2 リモート観測システム	2
第 3 章 仰角データ取得システムの改善	3
3.1 望遠鏡仰角データ取得システムの改善	3
3.1.1 角度情報データ取得の概要	3
3.1.2 仰角データ取得における問題点	6
3.1.3 PYNQ を用いた新システムの導入	9
3.2 望遠鏡への実装	11
3.2.1 新データ取得システムのインストール	11
3.2.2 同期信号取得の確認	12
3.2.3 同期信号の分配と仰角データ取得の確認	13
3.3 メンテナンスと安定運用	14
3.3.1 電源供給法の見直し	14

3.3.2 温度モニターの実装	14
第4章 検出器アライメントの較正	15
4.1 従来の検出器アライメントと問題点	15
4.1.1 スキャン軸に対する傾き	15
4.1.2 要求されるアライメント性能	15
4.1.3 視線軸方向まわりの回転による較正	15
4.2 月を用いた傾き角の算出	15
4.2.1 なぜ月なのか？	15
4.2.2 必要な回転角	15
4.2.3 回転する上でのジグの必要性	15
4.3 ジグの設計と現地インストール	15
4.3.1 固定用ジグの作成	15
4.3.2 望遠鏡への実装	15
4.4 天体を用いた較正結果の確認	15
4.4.1 月データによる確認	15
4.4.2 木星データによる確認	15
4.5 検出器間差分で見る大気揺らぎの抑制	15
4.5.1 なぜ差分をとるのか？	15
4.5.2 解析による確認	15
第5章 今後の展望	16
5.1 大気揺らぎに由来するノイズのモデリング	16
5.2 両偏波アンテナを搭載した焦点面検出器のアップデート	16
第6章 まとめ	17
第7章 謝辞	18
参考文献	19
付録A Zynqへのlinux搭載	20

第1章 序論

1.1 CMB の異方性と現代宇宙論

1.1.1 CMB の温度異方性

1.1.2 Λ -CDM モデル

1.1.3 地平線問題

1.2 CMB の偏光とインフレーション理論

1.2.1 インフレーション理論

1.2.2 CMB の偏光モード

1.2.3 偏光 B モードの探索状況

1.3 CMB の偏光とニュートリノ質量和

1.3.1 光学的厚み τ

1.3.2 ニュートリノ質量和との縮退

1.3.3 偏光 E モードと τ

第2章 GroundBIRD 実験

2.1 実験概要

2.1.1 GroundBIRD 望遠鏡

2.1.2 独創的なスキャン戦略

2.1.3 物理ターゲット

2.2 現在の観測状況

2.2.1 検出器のフルアレイインストール

2.2.2 リモート観測システム

第3章 仰角データ取得システムの改善

CMB 観測においては、検出器の時系列データと望遠鏡の角度データを途切れることなく連続的に取得し続けなければいけない。そのため、角度データ取得システムは安定的でかつ操作性がよいものであることが求められる。本章では既存の望遠鏡の仰角データ取得システムを改善し、その動作確認を行なった。

3.1 望遠鏡仰角データ取得システムの改善

3.1.1 角度情報データ取得の概要

はじめに、GroundBIRD 全体でのデータ取得システムの概要を述べる。CMB 観測においては検出器の信号を時系列データ（以下、TOD と略す）として取得する。最終的なマップ作成のためには、TOD と同期して望遠鏡の視線情報（角度データ）を取得することが求められる。GroundBIRD では望遠鏡の仰角方向と方位角方向で 2 つの角度データを取得している。連続回転する回転台の上部と下部は回転継手によって電気的に接続されている（図 3.1）。

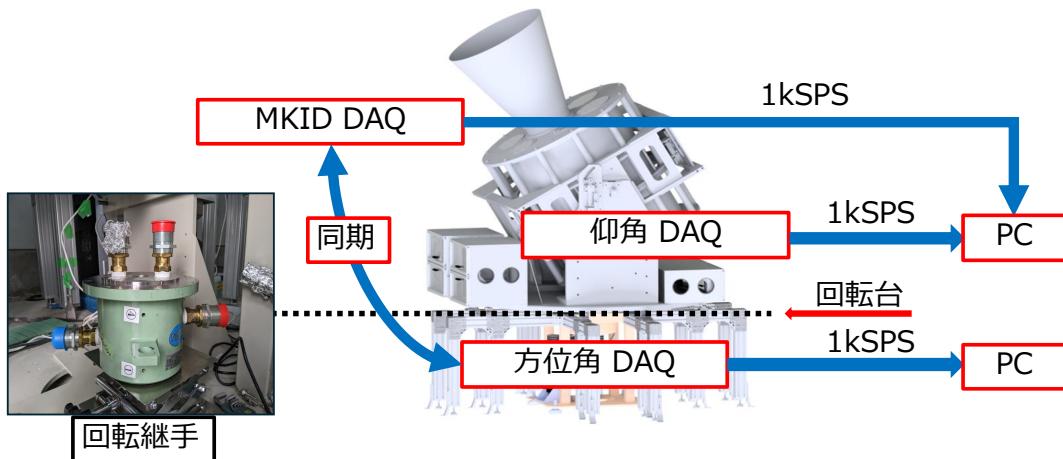


図 3.1: GroundBIRD の検出器データと角度データ取得系の概要。回転する回転台の上下での信号同期は“回転継手”が担っている。

仰角方向の角度データは、望遠鏡の側面に取り付けられたロータリーエンコーダー（Canon, R-1SL [1]）を使用し、Digilent 製の FPGA ボード Zybo Z7-20 ([2]、以下では単に Zybo と記す）で読み出す（図 3.2）。FPGA とは Field Programmable Gate Array の略で、様々な論

理回路がチップに搭載されており、使用者が配線を自由に組み合わせて論理回路を作ることができるデバイスである。FPGA では特定の演算を行う回路を作成できるため、高速処理を可能にする。また並列処理も得意である。エンコーダーは 4 秒角 ($1.1 \cdot 10^{-3}$ °) もの高い角度分解能を持つ。

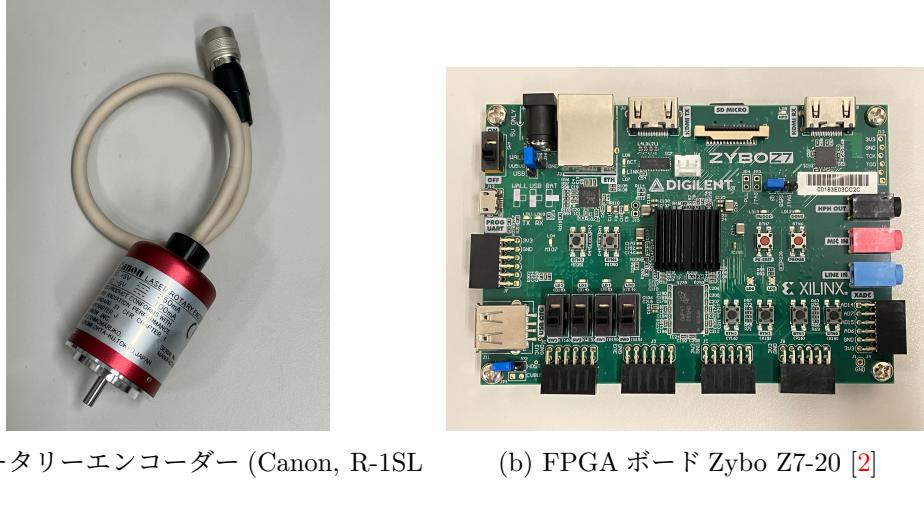


図 3.2: 仰角方向の角度データ読み出し

方位角方向の角度データは、回転台下部に取り付けられたロータリーエンコーダー (HEIDENHAIN, ERM220 [3]) を使用し、Xilinx 製の FPGA ボード Spartan3E [4] で読み出す (図 3.3)。エンコーダー自体の角度分解能は 2.6 分角 ($4.4 \cdot 10^{-2}$ °) である。さらに、平滑化フィルターを用いた方位角データの補完を行うことで、角度分解能を $5.7 \cdot 10^{-2}$ 分角 ($9.5 \cdot 10^{-4}$ °) に向上させている [5]。

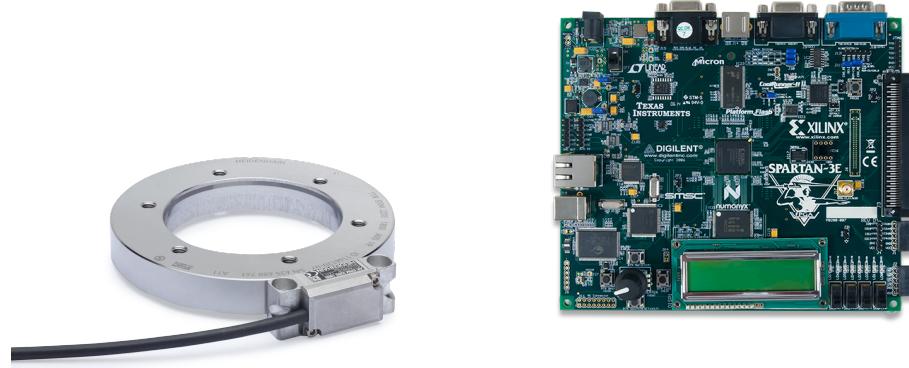


図 3.3: 方位角方向の角度データ読み出し

次に検出器のデータと方位角データに求められる同期精度を見積もる。時刻同期の精度を Δt とする。GroundBIRD の方位角方向の回転速度は最大で $120^\circ/s$ になる。方位角方向での角度分解能 $\Delta\phi$ は

$$\Delta\phi = 120^\circ/s \cdot \Delta t \quad (3.1)$$

になる。時刻同期による角度の決定精度がエンコーダーの角度分解能よりも十分小さいことを課す。時刻同期による角度決定精度をデータ補完によって向上したエンコーダーの角度分解能である $9.5 \cdot 10^{-4}^\circ$ の 1% 未満と要求すると、 Δt の上限は

$$\Delta t < \frac{9.5 \cdot 10^{-4}^\circ \cdot 0.01}{120^\circ/s} = 79\text{ns} \quad (3.2)$$

となる。この正確な時刻同期が必要になるため、仰角と方位角の角度データ読み出しで共に FPGA ボードを使用している。先行研究 [5] により、時刻同期精度を 55 ns に抑え、要求を満たす精度を実現している。

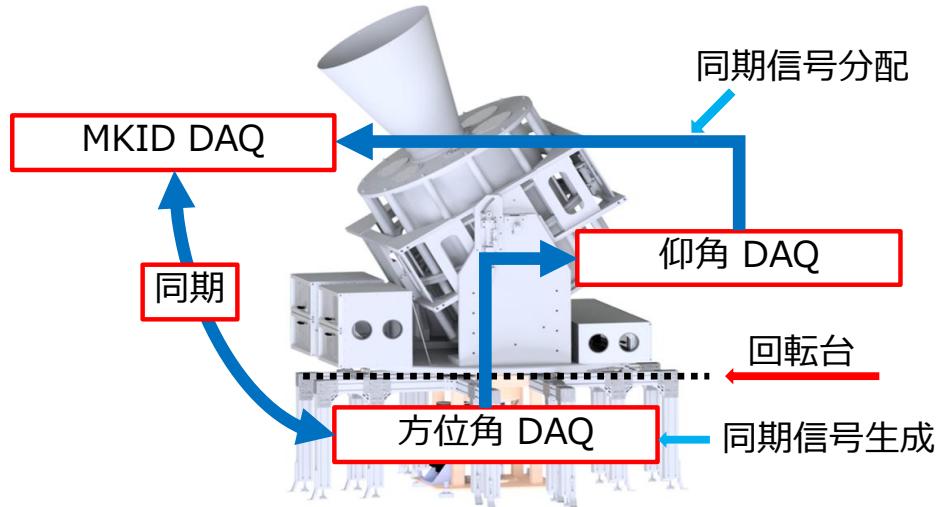


図 3.4: GroundBIRD の同期信号の流れ。回転台下部の方位角 DAQ ボードで生成された同期信号が回転継手を介して回転台上部の MKID の DAQ ボードに送信される。その際、仰角 DAQ ボードで同期信号を分配している。

GroundBIRD での同期信号の流れを説明する (図 3.4)。望遠鏡が連続回転するため、回転台の上下の電気的な接続に同軸ケーブルのような通常の信号線は使用できない。そのため、回転継手を介して回転台上下での信号を共有している。また、回転台の上下のデータ取得系でレートの遅い “同期信号” を回転継手を介して共有することで時刻の同期を図っている。

同期信号によるデータ同期を次のステップで行なっている。

1. 回転台下部の方位角 DAQ ボードから 1 秒に 1 回、基準となる同期信号を出力する。
2. 回転継手を介して回転台上部に届いた同期信号を仰角 DAQ ボードに入力する。
3. 回転台下部では、同期信号を出力した時刻情報を方位角のエンコーダーデータと共に保存する。
4. 仰角 DAQ ボード同期信号を分配し、検出器の DAQ ボードに送る。
5. 同期信号の到達した時刻情報を検出器の TOD と共に保存する。
6. 2 種類の TOD の時刻情報を用いて、各時刻での検出器信号と角度データの同期を行う。

3.1.2 仰角データ取得における問題点

本論文の研究対象である仰角のデータ取得に関して詳細に説明する。仰角のデータ取得系が担う役割は以下の2つである。

- 仰角の角度データを取得する
- 同期信号の分配

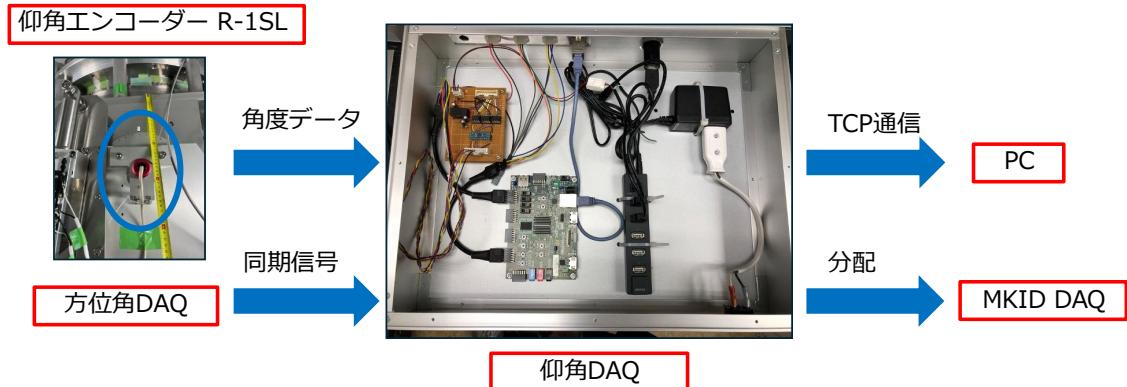


図 3.5: 仰角データ取得システムの全体像。信号処理のほぼ全てを Zybo [2] が担う。信号の電圧変換、Zybo、電源からなるコンパクトなデータ取得系は1つのボックス内に配置されている。

データ取得系は非常にコンパクトであり(図 3.5)、信号の処理は Zybo 内に搭載されている“Zynq [6]”と呼ばれるチップで行っている。Zynq は Xilinx が開発した、CPU、FPGA などが1チップに統合された System on Chip (SoC) の1つである。Zynq は CPU を搭載する Processing System (PS) 部分と、FPGA を搭載する Programmable Logic (PL) 部分に大きく分けられ、FPGA が得意とする並列処理や高速処理と、CPU が得意とする複雑な処理とで役割を分担できるため、効率の良い処理が可能となる。先行研究で FPGA に搭載するファームウェア¹の開発がなされており、実装されている。

FPGA 上でのファームウェアの全体図を図 3.6 に示す。設計には Vivado という Xilinx 製の開発ソフトを用いる。様々な IP コア(機能ごとの回路のまとめ)を IP インテグレータという GUI 上で配線し、全体のファームウェアを構成する。IP は自作の IP と、Xilinx 製の IP を組み合わせて使用している。

Zynq 内での処理の模式図を図 3.7 に示す。仰角エンコーダーはインクリメンタル方式を採用しており、エンコーダーからの出力信号は A 相、B 相、Z 相の3相からなる。エンコーダーからの出力パルス数で角度の変化量、A 相と B 相のパルスの立ち上がり順で回転方向が分かる仕様になっている。また、Z 相信号は1回転で1度出力され、回転の原点として使用される。3 相の信号は Zynq 内のデコーダー部分で角度情報として翻訳され、1kSPS で “FIFO [7]” に充填される。FIFO とは first-in first-out メモリのこと、データを格納し、取り出す

¹FPGA に組み込む機能を本論文ではファームウェアと呼ぶこととする。

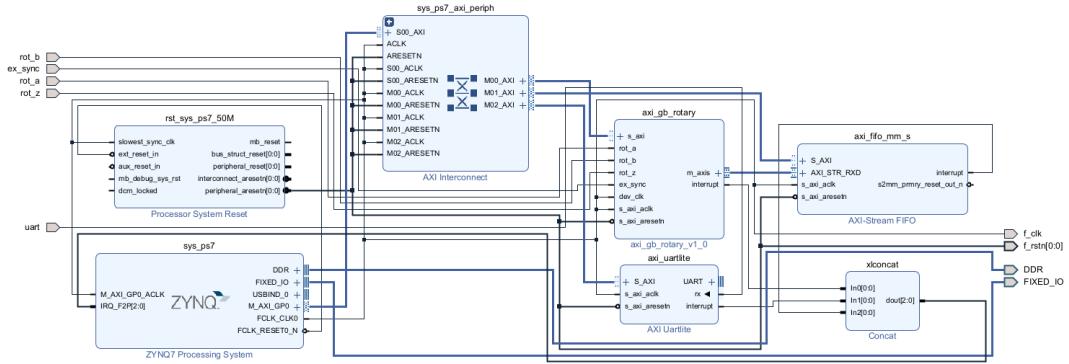


図 3.6: IP インテグレータの配線図。青く囲まれているブロックが IP コアを表す。ブロックの左側から出る線が入力信号で、右側から出る線が出力信号になっている。図中の “axi_gb_rotary” と呼ばれる IP が自作 IP で、デコーダーの役割を担っている。

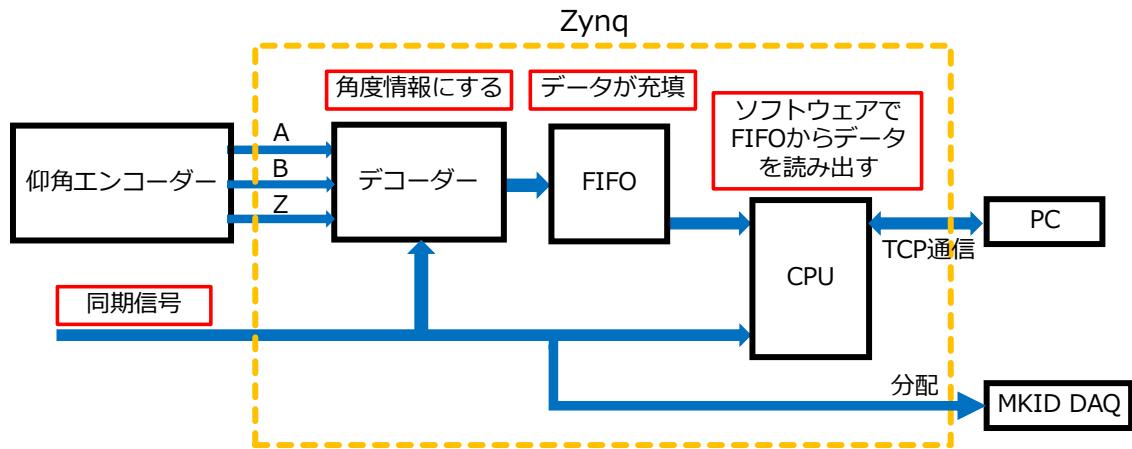


図 3.7: Zynq 内での信号処理。エンコーダーからの角度データ処理と方位角 DAQ ボードからの同期信号の分配を 1 チップ内で行う。

際は、格納した順番通りに先に格納したデータから取り出す構成になっている。FIFO に格納されたデータは CPU のソフトウェアで読み出し、TCP 通信で PC へと送信される。

同期信号の通信方式は UART²を使用しており、デコーダー部分でタイムスタンプ(測定開始時からの経過時間)情報を取得する。また、MKID の DAQ は複数のボードを使用するため、同期信号を分配させ、送信している。

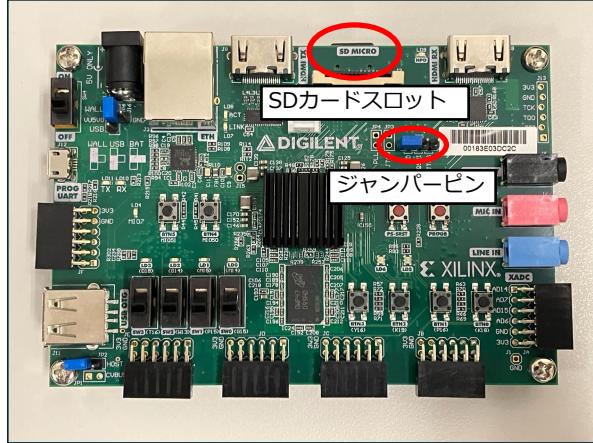


図 3.8: Zybo には SD カードスロットが付いている。システムを起動するために必要なファイルを SD カードに書き込み、スロットに差し込んで電源を入れることで稼働が開始する。ジャンパーピンは SD に設定しておく。

以上の構成で仰角 DAQ システムを稼働させていた(図 3.8)が、観測の長期運用を考えた際に、問題点を抱えていた。Zynq でのデータ処理と PC への通信を OS のない(ペアメタル³)環境でソフトウェアを動かすことで行っている。このことにより、システム全体が硬直的になっており、その扱いにおいて柔軟性がない。柔軟性がないことで起きる問題として以下のものが挙げられる。

- データ取得が異常終了した際のメンテナンスが難しい
- OS が介さない通信による信頼性の低下
- ソフトウェア、FPGA 面での改良が難しい

最も大きな問題はメンテナンス性である。観測の有無に問わず、望遠鏡の角度データは常に取得し続けており、安定した仰角データ取得が必要不可欠である。しかし、仰角エンコーダーから PC までのデータの流れが途切れてしまうことがある。それは、通信のエラーや停電等による電源系統の不具合など様々な要因からくるもので、長期運用をする上ではある程度避けられないものである。その際、エンコーダーと Zybo 間の通信が切れて、Zynq 内で

²UART 通信では、送信側と受信側で通信速度を決めておき、1byte (8bits) ずつ情報のやり取りを行う。1byte のまとめでは、“start bit (1bit)”、“data (8bits)”、“parity bit (1bit、情報の誤検知に使用)”、“stop bit (1bit)” の 4 つで構成され、全 11bits になる。

³本来は「剥き出しの金属」という意味だが、転じて OS がインストールされていないコンピュータのことを指す。

のエンコーダー情報が失われる。加えて、エンコーダーの原点情報も失われるため、通信を再開させた際に読み出した角度の値にオフセット値が乗ってしまう。Zynq 内で原点情報を記憶させるには望遠鏡の仰角を動かし、エンコーダーの Z 相信号を取得して値をリセットすることが必要であり、手間のかかる工程になる。この一連のメンテナンスをまとめると以下のステップに分けられる。

1. Zybo を再起動させて再度ソフトウェアを動かす
2. 望遠鏡の仰角を動かして Z 相信号を入力し、原点情報を記憶
3. 仰角を 70° まで動かして固定

この中で、Zybo の再起動と仰角を動かす際にケーブルに変な張力がかかっていないかの目視に現地での作業を要する。リモートでの望遠鏡運用を進める上で、少しでも現地で必要な作業を減らし、リモートでメンテナンスができることが望まれる。

また、Zybo と PC との TCP 通信を行うために、ベアメタル環境では TCP/IP のプロトコルスタックを独自で実装しなければいけない。今回は “lwIP(lightweight IP)” と呼ばれるオープンソースの TCP/IP プロトコルスタックを使用して TCP 通信を実装している。そのため、通信に関わるソフトウェアが複雑化する上、OS が通信を取り仕切るよりも信頼性に欠ける。

加えて、システムに組み込まれたソフトウェアや FPGA のファームウェアは固定化されており、今後の運用で変更点が生じた時に、SD カードに新しいファイルシステムを書き込んで全面的にシステム更新しなければならず、労力を要する。システムのカスタマイズ性を上げ、変更を容易にできるようになれば、運用に関わるコストを削減することができる。

3.1.3 PYNQ を用いた新システムの導入

上に述べた問題を改善するために、本研究では既存の Zynq システムに OS を搭載し、ソフトウェアを OS 上で動かすことによってシステム全体の柔軟性向上を試みた。これにより上記の問題点に対して

- データ取得が異常終了した際のメンテナンスをリモート主体で行える
- OS が通信を介することで信頼性が向上
- ソフトウェア、FPGA 面での改良をシステムを起動したまま行える

という改善が期待できる。一方でベアメタル環境に比べて OS をインストールすることで実行時間と使用メモリが増えるが、FIFO へのデータ充填と FIFO からの読み出しあとはともに 1kSPS であり、FIFO の容量も十分であるため、性能に問題は出ない。Zynq に搭載する OS は基本的に linux ベースであるが、今回は “PYNQ [8]” と呼ばれる Ubuntu⁴、Jupyter および Python をベースとしたフレームワークを搭載した。PYNQ を採用した理由として

⁴ 今回は Ubuntu 22.04 がベースになっている

- ソフトウェアを PYNQ 上の Python スクリプトで動かせるため、通信スクリプトを簡潔に記述できる
- システム起動に必要なブートイメージファイルの作成が容易
- “Overlay” と呼ばれる機能を使用することで FPGA のファームウェアを Python で容易に変更できる

が挙げられる。

ここで PYNQ のイメージファイル作成の概要を説明する。基本的な手順は [9] を参考にし、作業環境は Docker 上の Ubuntu 20.04 で構築した。その際に使用した開発ツールとバージョンを表 3.1 にまとめる

表 3.1: PYNQ イメージ作成で使用したツールとバージョン

ツール	バージョン
Vivado	2022.1
PYNQ linux イメージ	3.0.1
PetaLinux	2022.1

PetaLinux とは Xilinx が提供する、Zynq をはじめとする SoC 用の linux システムをビルドするためのツールである。この環境のもとで以下のステップで PYNQ イメージを作成した。

1. ベースとなる FPGA ファームウェアの作成
2. Zybo 用のスペックファイルの作成
3. ビルド

FPGA ファームウェアは Overlay によって Zybo 起動時に変更できるため、この時点では最も単純な回路を Vivado で準備してやれば良い。それをもとに Vivado 上でコンパイルをし、生成された 4 つの回路情報を持つファイル (.xsa ファイル、.bit ファイル、.hwh ファイル、.tcl ファイル) を取得する。

スペックファイルは Zybo のスペック情報を.spec ファイルとして作成する。その後、作成した全ファイルをビルド用のディレクトリに置く。最後に make を実行してビルドを行い、PYNQ イメージファイル (.img ファイル) を生成して完了する。

このイメージファイルを SD カードに dd コマンド等で書き込み、Zybo の SD カードスロットに差し込むことで PYNQ が起動する。また、起動時に Overlay スクリプトを走らせて図 3.6 で示したファームウェアで FPGA の回路を上書きするように設定した。これにより、本来の回路情報が PYNQ 上で再現される。Overlay を用いて FPGA のファームウェアを変更する仕組みを図 3.9 に示す。

Overlay に必要なファイルは Vivado でコンパイルをして生成される.bit ファイルと.hwh ファイルの 2 つである。これらは同じディレクトリに置いておく。Python で Overlay クラスをインポートし、.bit ファイルを読み込ませることで Overlay は実行される。その際、Overlay クラスは同じディレクトリにある.hwh ファイルも読み込んでくれる。図で示した



図 3.9: PYNQ から FPGA のファームウェアを変更する模式図。Vivado で設計した新しい回路ファイルを Overlay クラスに読み込ませることで容易に変更ができる。

ように Overlay スクリプトはたった 2 行で書くことができ、ファームウェアの変更は非常に容易に実行できる。

最後に、Zynq 内で動かすスクリプトを PYNQ 用に Python で再構築した。

3.2 望遠鏡への実装

3.2.1 新データ取得システムのインストール

新しく導入した仰角データ取得システムを、自分の手元でできる動作確認をした後に実際の望遠鏡に実装、という流れでインストールした。

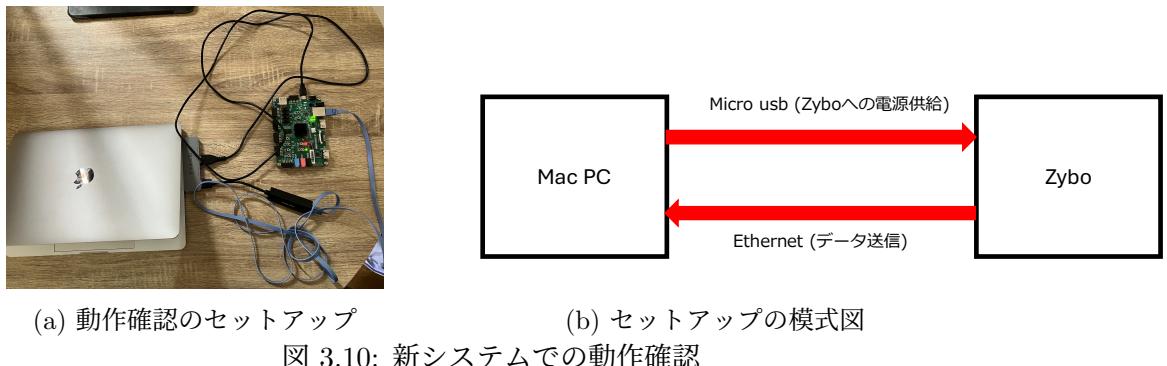


図 3.10: 新システムでの動作確認

動作確認を、図 3.10 に示したセットアップで行なった。ロータリーエンコーダーとは接続せず、同期信号も送らない単純なものである。図 3.10 の左図にあるように、Zybo の電源を入れ PYNQ が起動するとボード右側の “DONE” ランプが緑色に点灯する。自分の Mac PC をデータ取得 PC として、Micro usb で Zybo への電源供給を行い、データ通信は Ethernet ケーブルで行った。PC 上でデータ読み出しのスクリプトを動かして、Zynq の挙動に問題がないかをチェックした。期待される挙動は

- Zynq が正しく機能し、PC 側でデータを読み出せる
- エンコーダーデータは 0 として取得されている

- タイムスタンプは時間とともに加算されている

の3点である。データの通信が始まるとZyboのEthernetコネクタ付近のランプが点滅し、PC側でのデータ読み出しが正常に動くことを確認した。PCに保存したデータファイルをチェックし、期待されるデータを取得できていることも確認した。取得したエンコーダーデータとタイムスタンプデータのプロットを図3.11に示す。プロットの横軸はデータのサンプリングナンバー(1kSPS)を表す。

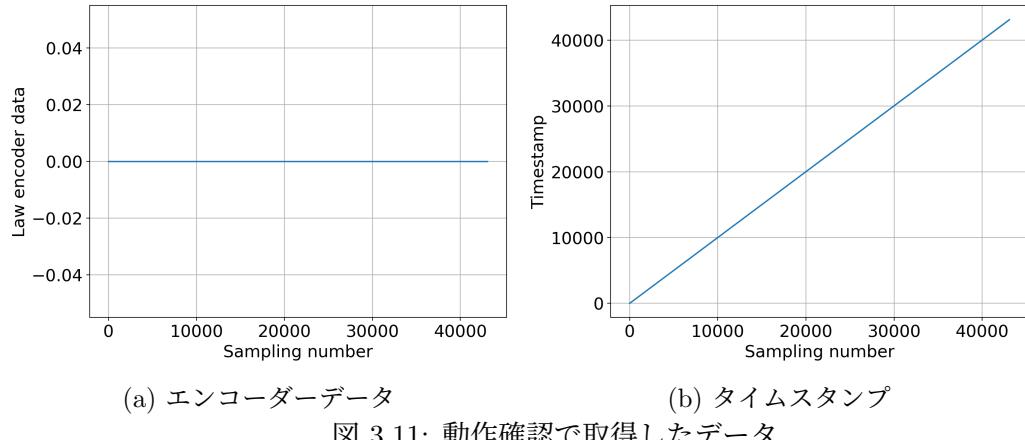


図 3.11: 動作確認で取得したデータ

以上の結果から新システムの挙動が問題ないと判断し、次に実際の望遠鏡にインストールした。インストールはSDカードを新システムのものに交換するだけで完了し、PYNQの起動を確認した。

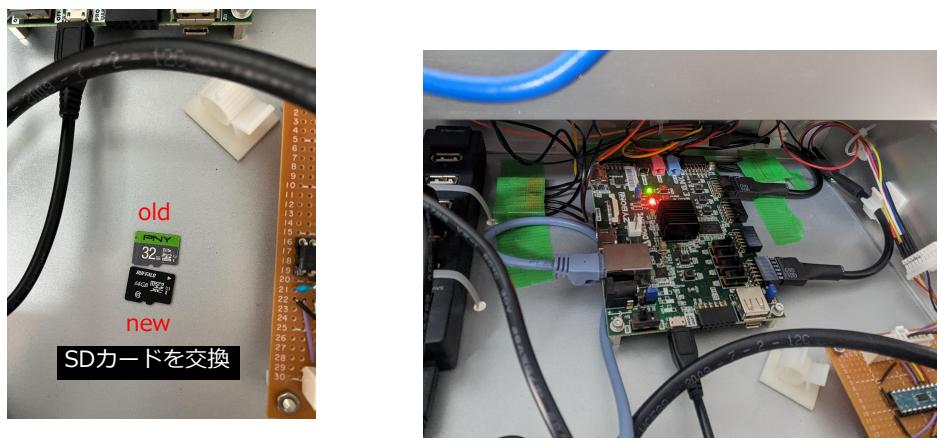


図 3.12: 新システムのインストール

3.2.2 同期信号取得の確認

インストール後に実際の望遠鏡システムでデータを読み出せるかの動作確認を行なった。まず、方位角DAQからの同期信号を正しく取得して仰角データとして保存できているのかを確認した。結果を図3.13に示す。同期信号は1秒に1回出力されるので、仰角データで

は同期信号の番号(図では“Sync_id”と記す)が1秒で1ずつ増加する形で見えるはずであり、その結果を確認することができた。

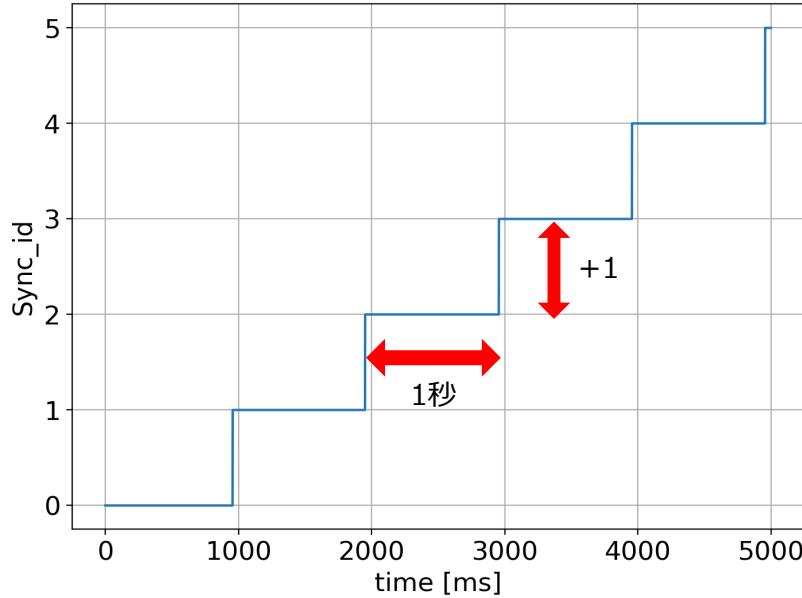


図 3.13: 仰角データからみた同期信号。方位角 DAQ からの同期信号を取得するごとに“Sync_id”が1ずつ加算される。

3.2.3 同期信号の分配と仰角データ取得の確認

次に、取得した同期信号を MKID DAQ に分配できていることと仰角データを正しく読み出せているかを確認した。一度失ったエンコーダーの原点情報を再度取得するためにも、望遠鏡の仰角を 90° と 70° の間で何度か動かして、さらに並行して MKID のデータも取得した。それらのデータを使って確認を行なった。確認の手順は以下である。

1. テスト用として取った MKID データを読み出す
2. 正しく動作していれば MKID データに同期した時間情報と仰角データを取得できる
3. その仰角データが正しい値を読み出しているかを確認

結果を図 3.14 に示す。MKID データから同期情報を取得でき、読み出した仰角データが 90° と 70° の間で正しく動いていることも確認した。

以上から新システムが実際の望遠鏡で問題なく動作することを確認した。その後、動作が安定して長期間行われるかをチェックした。

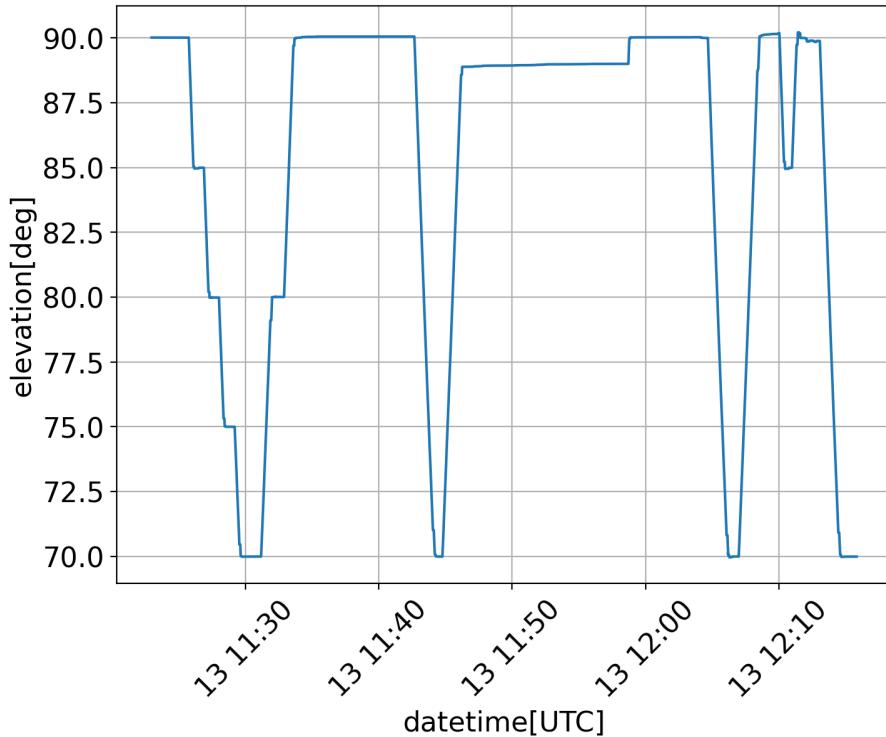


図 3.14: 読み出した仰角データ。横軸 (2024/3/13 の UTC 時間) の時間情報が実際の作業時間とリンクしており、MKID データで同期信号が正しく取得できていることを反映している。

3.3 メンテナンスと安定運用

3.3.1 動作の不安定性

インストール後、動作の安定性に問題があり、データ取得が途切れることが何度も発生した。途切れる原因は Zybo の電源が一時的に落ちることによるものであった。インストール時の動作確認で問題がなかったことから、Zynq 内でのデータ処理と通信自体に問題がある可能性はないと考えた。他に原因となりうるものは

- OS が搭載されたことで Zybo の消費電力が上がり、一時的に電源供給量が足りなくなる
- Zynq での消費電力も上がり、Zynq の温度が許容値よりも高くなってしまう
- そもそも Zybo のボード自体がどこかで劣化している

が挙げられる。3 つ目に関しては、経年劣化や落雷による停電時にダメージを受けたことなどが考えられるが、リモートからボード自体の性能を評価することが難しいため、まずは 1 つ目と 2 つ目の原因について調査した。

3.3.2 電源供給法の見直し

3.3.3 温度モニターの実装

第4章 検出器アライメントの較正

4.1 従来の検出器アライメントと問題点

4.1.1 スキャン軸に対する傾き

4.1.2 要求されるアライメント性能

4.1.3 視線軸方向まわりの回転による較正

4.2 月を用いた傾き角の算出

4.2.1 なぜ月なのか？

4.2.2 必要な回転角

4.2.3 回転する上でのジグの必要性

4.3 ジグの設計と現地インストール

4.3.1 固定用ジグの作成

4.3.2 望遠鏡への実装

4.4 天体を用いた較正結果の確認

4.4.1 月データによる確認

4.4.2 木星データによる確認

4.5 検出器間差分で見る大気揺らぎの抑制

4.5.1 なぜ差分をとるのか？

4.5.2 解析による確認

第5章 今後の展望

- 5.1 大気揺らぎに由来するノイズのモデリング
- 5.2 両偏波アンテナを搭載した焦点面検出器のアップデート

第6章 まとめ

matome

第7章 謝辞

ありがとうありがとう

参考文献

- [1] <https://canon.jp/biz/product/indtech/incremental-encoder/lineup/r1sl>
- [2] <https://digilent.com/reference/programmable-logic/zybo/start?redirect=1>
- [3] <https://www.heidenhain.co.jp/製品/角度エンコーダ/組込み型角度エンコーダ/erm-2000 シリーズ>
- [4] <https://japan.xilinx.com/support/documentation-navigation/silicon-devices/mature-products/spartan-3e.html>
- [5] 池満拓司. CMB 望遠鏡のデータ読み出しシステムの時刻同期と較正に関する開発研究. 京都大学理学研究科 修士論文 2020.
- [6] <https://docs.amd.com/v/u/en-US/ds187-XC7Z010-XC7Z020-Data-Sheet>
- [7] https://japan.xilinx.com/products/intellectual-property/axi_fifo.html
- [8] <http://www.pynq.io>
- [9] <https://wasa-labo.com/wp/?p=1102>

付録A Zynqへのlinux搭載

時間あつたら書きたい