修士論文

ASTRO-H 衛星における時刻付けシステムの開発

埼玉大学 理工学研究科 物理機能系専攻 物理学コース
田代・寺田研究室
博士前期課程 2 年

神頭 知美

提出: 2010 年 2 月 5 日

概要

近年、人工衛星・科学衛星の大型化に伴い、開発にかかるコストが莫大になっている。これは、衛星ごとにデータ収集システムを専用設計で構築している事も一因である。そこで世界的に衛星組込ネットワークを標準化し、どの衛星もおなじネットワーク・プロトコルを使うことで、検証時間を短縮し、知識や資産の継承を可能とする試みがなされている。現在、このようなネットワーク・プロトコルのなかで最も標準化が進んでいるものが「SpaceWire」である。次期 X 線天文衛星 ASTRO-H は、日本で初めて SpaceWire を本格的に採用する大型科学衛星で、パルサーなど速い時間変動をする X 線天体を観測するために、高い時刻性能 (〜10 マイクロ秒) が求められている。現在、ASTRO-H で計画されている SpaceWire を用いた衛星時刻 (TI) の配信方法のなかで、特に秒以下の TI は Time-Code とよばれる SpaceWire の最優先時刻コードで配信される。しかし、Time-Code の刻みは 15.6 ミリ秒しかなく、TI だけで要求時刻性能を達成することはできない。

そこで我々は、ASTRO-H 衛星でより細かい時刻をつける方法を検証し、検出器エレクトロニクスでフリーランクロックと同期したより細かい刻みの時刻 (LocalTime) と TI とを比較し内挿するという手法を提案した。さらに、実際に SpaceWire 搭載エレクトロニクスを用いて、この手法による時刻性能を詳細に確かめた。この手法では、(A)Time-Code のジッタ (時刻不定性) と、(B) フリーランクロック (水晶発振器) の温度に対する時刻安定度が、時刻精度に影響する。実測した結果、(A)Time-Code のジッタは、ASTRO-H 衛星のリンクレート (20~50 MHz) の場合、およそ数マイクロ秒程度であった。また、(B) フリーランクロックの時刻安定度は、温度安定度がおよそ ± 5 ℃の環境では 1 日でおよそ 10^{-7} 秒/秒であったが、TI と LocalTime の比較照合データを細かく (およそ 10 秒以下の周期で) 取ればほぼ無視できることがわかった。以上の実験結果から、我々の提案した手法で ASTRO-H に要求されている時刻性能が達成できることがわかった。

目次

第 1 章　ASTRO-H 衛星の時刻性能目標 1
1.1　X線天文学と時刻 . . . 1
1.2　ASTRO-H 衛星の概要 . . . 3
1.3　時刻性能の目標と要求 . . . 5
1.4　本論文の目的 . . . 6

第 2 章　次世代衛星組込ネットワーク SpaceWire 7
2.1　SpaceWire . . . 7
2.2　Time-Code . . . 10
2.3　Remote Memory Access Protocol (RMAP) . . . 13

第 3 章　ASTRO-H 搭載 時刻配信システム 15
3.1　衛星内時刻配信の概念 . . . 15
3.2　ASTRO-H における時刻配信 . . . 16
3.2.1　衛星のネットワーク構成 . . . 16
3.2.2　衛星時刻 (TI) の配信 . . . 17
3.3　時刻目標達成への課題 . . . 18

第 4 章　高時刻精度を持つイベント時刻付けシステムの概念設計 19
4.1　高分解能時刻を達成する方法 . . . 19
4.1.1　PLL による衛星時刻クロックへの同期を用いる方式 . . . 19
4.1.2　非同期クロックを用いた時刻 tick 内挿法 . . . 21
4.2　非同期クロックを用いた時刻 tick 内挿法における時刻性能 . . . 23
4.2.1　Time-Code のジッタ . . . 23
4.2.2　LocalTime の時刻安定度 . . . 25

第 5 章　SpaceWire 搭載機器への実装 27
5.1　実装方法の概要 . . . 27
5.1.1　Field Programmable Gate Array (FPGA) . . . 27
5.1.2　ハードウェア記述言語 . . . 28
5.2　実験機器 . . . 28
5.2.1　SpaceWire Digital I/O board . . . 29
5.2.2　SpaceCube 1 . . . 29
5.3　実装した機能 . . . 32
5.3.1　Time Master 機能 . . . 32
5.3.2　User Node における時刻付け機能 . . . 32

5.4 時刻付けに用いる各機能の動作確認実験 . . . 35
5.4.1 実験セットアップ . . . 35
5.4.2 結果 . . . 36

第 6 章 時刻精度の測定 39
6.1 Time-Code のジッタ測定 . . . 39
6.1.1 概要 . . . 39
6.1.2 実験セットアップと実験方法 . . . 39
6.1.3 結果 . . . 42
6.1.4 考察 . . . 42
6.2 時刻安定度の測定 . . . 45
6.2.1 概要 . . . 45
6.2.2 実験セットアップと実験方法 . . . 45
6.2.3 結果 1:常温下での時刻安定度 . . . 46
6.2.4 結果 2:一定温度での時刻安定度 . . . 48
6.2.5 考察 . . . 49
6.3 TI と LocalTime のサンプル周期と時刻安定度の測定 . . . 50
6.3.1 概要 . . . 50
6.3.2 方法 . . . 50
6.3.3 実験セットアップと実験方法 . . . 50
6.3.4 結果 . . . 51
6.3.5 考察 . . . 51
6.4 本章の結論 . . . 52

第 7 章 まとめと今後 53

付録 A 水晶発振器のドリフトの確認実験 55
A.1 実験セットアップと方法 . . . 55
A.2 結果 . . . 56
A.3 考察 . . . 56

付録 B イベント時刻付け機能の確認 57
B.1 セットアップ . . . 57
B.2 結果 . . . 58
B.3 イベント時刻付けツールのソース・コード . . . 59

付録 C Time Master の User FPGA の VHDL コード 61

付録 D TI と LocalTime のサンプル周期と時刻安定度の測定: 常温下・ルーターあり 65
D.1 概要 . . . 65
D.2 実験セットアップと実験方法 . . . 65
D.3 結果 . . . 65
D.4 考察 . . . 65

参考文献 69

謝辞 71

図目次

1.1 電磁波の波長と大気を透過率 1
1.2 X線天体の活動のタイムスケール 2
1.3 ASTRO-H 衛星の完成予想イメージ 3
1.4 「すざく」衛星搭載 HXD で検出した、かにパルサーの光度曲線 5

2.1 SpaceWire 規定による、Data Character、Control Character, Control Code(NULL,Time-Code) の定義 8
2.2 SpaceWire 規定によるパケットの型の定義 8
2.3 Time-Code の配信の仕組み 11
2.4 2 ノード間における Time-Code 配信 12
2.5 円環状ネットワークにおける Time-Code 配信 12
2.6 Time-Code が失われた時 12
2.7 SpaceWire レイヤと RMAP レイヤの関係 13
2.8 RMAP 通信のイメージ図 14
2.9 ASTRO-H におけるネットワーク構造のイメージと SMU の位置付 14

3.1 ASTRO-H 衛星のネットワーク構成 16
3.2 各機器における処理の概要 17
3.3 衛星時刻 (TI) のフォーマット 17

4.1 基本的な Phase Locked Loop 回路 20
4.2 基本的な Phase Locked Loop 回路の応答 20
4.3 Phase Locked Loop に分周器を入れた回路 21
4.4 非同期クロックを用いた時刻 tick 内挿法での Time Master と User Node の動作概要 . 22
4.5 時刻 tick 内挿法における MIO board からのテレメトリ出力 22
4.6 具体的な 時刻 tick 内挿法で高い時刻精度を得る方法 23
4.7 Time-Code のジッタが発生する原因 24
4.8 NULL 通信時の Time-Code ジッタの確率分布シミュレーション 25
4.9 LocalTime の安定度と時刻精度 26

5.1 自動化されたデジタル設計のイメージ 28
5.2 SpaceWire Digital I/O board の写真と各搭載エレクトロニクス 30
5.3 SpaceWire Digital I/O board のブロックダイアグラム 30
5.4 SpaceCube 1 の写真 31
5.5 SpaceCube 1 のブロックダイアグラム 31
5.6 SpaceWire Digital I/O board に実装したモジュールのブロック・ダイアグラム 34

5.7 確認用の実験セットアップ 35
5.8 Time Master の Tick と Time-Code の出力タイミングの確認 36
5.9 Time Master における Time-Code のカウントアップ確認 37
5.10 Time Master と User Node の Time-Code 比較 37
5.11 TI と LocalTime の照合データを SDRAM から読みだしたところ 38

6.1 実験セットアップの写真 40
6.2 Time-Code のジッタ測定セットアップ 40
6.3 GPSR : Furuno GF8050 と GPS アンテナ : Furuno AU-117A 41
6.4 ある日の GPS 衛星の軌道解析 41
6.5 ジッタ測定の結果 43
6.6 時刻安定度測定実験のセットアップ 45
6.7 ある日の常温下での時刻安定度 46
6.8 別の日に測定した常温下での時刻安定度 47
6.9 一定温度での時刻安定度 48
6.10 LocalTime の時刻安定度と温度変動値の相関 51

A.1 階段状温度変動での実験における恒温槽内の温度変動 55
A.2 階段状温度変動での時刻安定度 56

B.1 イベント時刻付け機能の確認実験セットアップ 57

D.1 hop なしにおけるサンプル周期と時刻安定度 66
D.2 1 hop におけるサンプル周期と時刻安定度 66
D.3 サンプル周期とその間の時刻安定度の相関 67

表目次

1.1 ASTRO-H 搭載予定の観測機器の性能 4

3.1 TI の時刻範囲と配信方法のまとめ 17

5.1 SpaceWire Digital I/O board の性能 30
5.2 SpaceCube 1 の性能 31

6.1 遅延時間とジッタの測定値 42
6.2 時刻安定度の測定結果のまとめ 49

B.1 イベントの時刻と前後の時刻 HK データ 58

第 1 章

ASTRO-H 衛星の時刻性能目標

1.1 X線天文学と時刻

1609 年に世界で初めて望遠鏡を夜空に向けて天体観測を行ったのは、イタリア人のガリレオ・ガリレイだと言われる。以来、300 年余りにわたって、望遠鏡といえば「可視光」の光学望遠鏡のことを指しており、まして天体からX線が放射されていることなど誰も知らなかった。地球の大気がX線を含む放射線を遮断するため、地上からX線をつかった天体観測をすることができないからであった (図 1.1)。しかし 20 世紀に入って大気圏外に衛星やロケットで検出器を運ぶことができるようになると、宇宙のさまざまな天体からX線が放射されていることがわかってきた。X線は波長が短く、エネルギーの高い光子なので、高エネルギーの天体現象を見ることができる。例えば、高エネルギーまで加速された電子などの粒子が放射する電磁波として、強い磁場の中を通る場合にシンクロトロン放射が天体から放射されたり、星間空間に満ちた 1 億度もの高温プラズマからは熱的分布をもつ電子からの制動放射が放出される。さらに、ブラックホールの周囲にできた降着円盤は光速に近いガスのジェットを噴出する。X線はこうした高エネルギー天体の観測に適している。宇宙の大部分はX線でなければ観測できないとも言われている。いまやX線天文学は、電波、可視光、赤外線天文学などと並んで、天体現象の解明に必要不可欠な学問である。

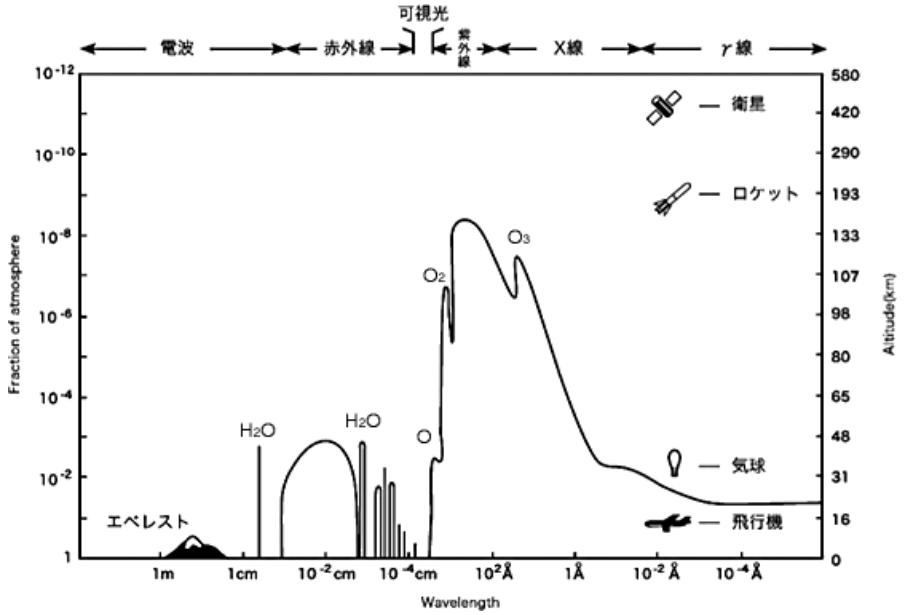

図 1.1　電磁波の波長と大気を透過率。縦軸:電磁波が吸収される高度、横軸:電磁波の波長。参考のため、吸収する分子と高度ごとにある物も載せてある。(ASTRO-H プロジェクトサイト [http://astro-h.isas.jaxa.jp/] より作成)

X線で宇宙を見ると、幅広いタイムスケールで時間変動する天体を観測することができる。図 1.2 に、X線を放射する天体のタイムスケールをまとめた。その中でも最も速い時間変動をする天体に「パルサー」がある。パルサーは最も速いもので数ミリ秒という超高速な自転をしており、この自転にともなった周期的パルスが観測される。このような超高速回転のできる天体はたいへんコンパクトでないとならず、理論的に半径 10km 程度の「中性子星」だと考えられている。中性子星は、太陽の 20 倍ほどの質量をもつ恒星の最期・超新星爆発のあとに残される星の「芯」が、爆発の反動によって押し固められたものである。あまりに高密度になりすぎ原子の形をとどめることができず、電子が陽子に捕獲されて中性子ばかりになっている。中性子星パルサーの周期的パルスは、超高速自転にともなう磁気双極子放射が起源と考えられており、見積もられる磁場は 10^8〜10^9 テスラである。地球上でもっとも強力な永久磁石「ネオジム磁石」の磁場：およそ 0.5 テスラと比較すると、その強さがわかる。中性子星パルサーの強力な磁場中では電子などの荷電粒子が高エネルギーにまで加速されていて、電波や可視光、X線、ガンマ線といった電磁波を放射している。このように中性子星パルサーは、我々にとって地上で実現できない極限物理の実験場でもある。

パルサーなどの速い時間変動をする天体を観測するため、X線天文衛星には、高い感度だけでなく高い時刻性能が要求される。2010 年現在稼働中の日本のX線天文衛星「すざく」 [1] 搭載 硬X線検出器 (Hard X-ray Detector：HXD) [2][3] は、2010 年現在、10-600 keV の光子エネルギーで世界最高感度を誇る。それだけでなく、高い時間分解能 (61 μsec) をもつ [4] ので、数多くの中性子星パルサーの観測にも使われてきた。しかし、パルサーは、電波で検出した個数 (およそ 1500 個) [5] と比べて、X線で検出した個数 (およそ 150 個) が非常に少ない。理由として、もとからX線パルサーの数が少ない、X線検出器の感度が足りずに見つけられていない、のふたつが考えられるが、まだ高い時間分解能と高い感度の両方を兼ね備えた検出器が少なく、どちらかはよくわかっていない。

2014 年打ち上げを目指すX線天文衛星 ASTRO-H は、「すざく」HXD を上回る過去最高感度でいままで発見できなかったパルサーの発見も期待されている。こうした発見が、いままで解くことのできなかったパルサーなどの高エネルギー天体での極限状態の物理を明らかにするかもしれない。

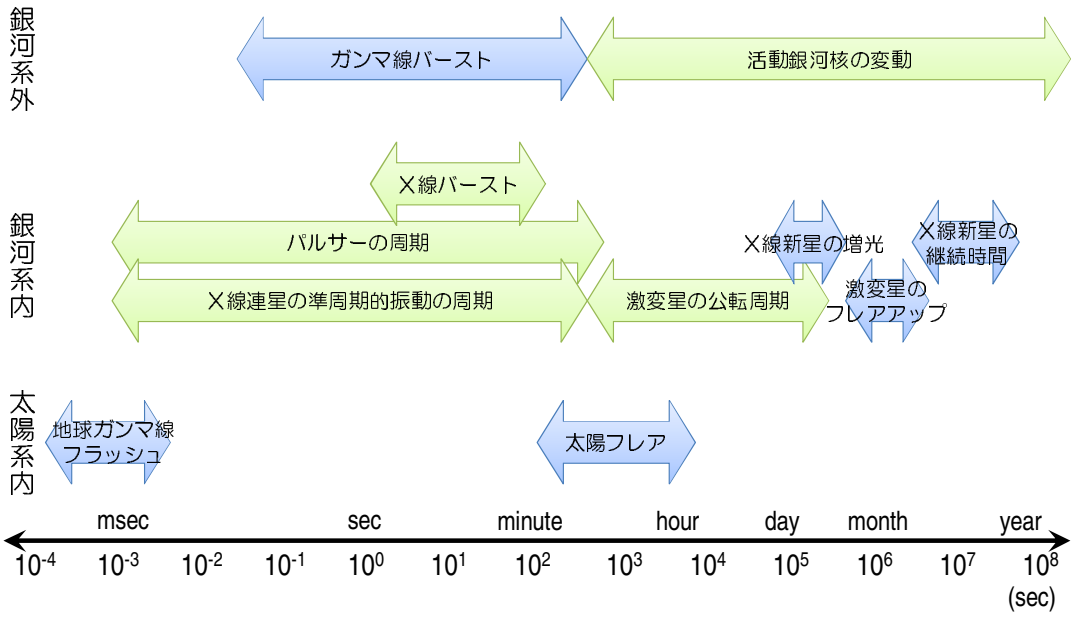

図 1.2　X線天体の活動のタイムスケール。緑は定常現象、青は突発現象をさす。

1.2 ASTRO-H 衛星の概要

ASTRO-H [6] は、2014 年打ち上げを目指している日本第 6 番目のX線天文衛星である。JAXA[*1] を中心とした日本の研究機関・大学だけでなく、NASA[*2]、ESA[*3] などの国際協力で開発が進んでいる。科学的目標として、前述のように高感度観測で新しいX線源の発見するだけでなく、高いエネルギー分解能の分光観測でブラックホールの回転運動の様子を解明すること、広視野・広帯域の撮像で超新星残骸での宇宙線加速の様子を明らかにすること、などが期待されている。

以下に、ASTRO-H に搭載される予定の観測機器の概要と性能をまとめる。

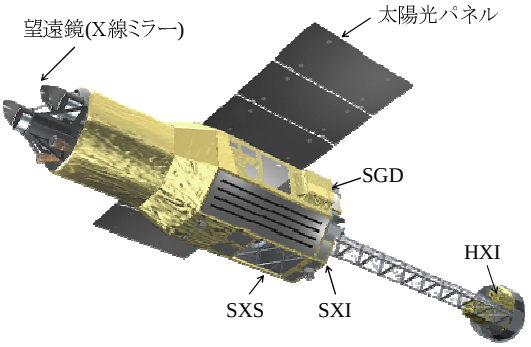

図 1.3 ASTRO-H 衛星の完成予想イメージ©JAXA

SXS (Soft X-ray Spectrometer) [7] + **SXT-S (Soft X-ray Telescope)**
(軟X線分光システム)

SXS は、X線マイクロカロリメーターと呼ばれる装置で、X線をあびた吸収体の温度上昇からX線エネルギーを測定する。X線のエネルギーを光子で検出するタイプの装置なので、キャリアの統計揺らぎが抑えられ、 eV 程度のエネルギー分解能が達成される。これまで同帯域で分光観測に使われてきたX線用 Charged-Coupled Device (CCD) と比べて 10 倍以上のエネルギー分解能に相当する。SXS のエネルギー帯域のX線は、屈折角が大きいバウムクーヘン型 (Walter II 型) の多層ミラー (SXT) を使用し全反射で集光される。焦点距離は 6 メートルである。

SXI (Soft X-ray Imager) [8][9] + **SXT-I (Soft X-ray Telescope)**
(軟X線撮像システム)

SXI は、X線 CCD カメラである。「すざく」搭載 XIS (X-ray Imaging Spectrometer) と同じ MOS 型 CCD であるが、電荷損失を防ぐため SXI は P チャネルを採用している。背面照射型にすることで電極による吸収がなくなり、より低いエネルギー帯域のX線光子に高い感度を持つ。SXS と比べて、広視野 (35 × 35 分角) かつ広帯域 (0.3 – 12 keV) でX線画像を撮影することができる。ASTRO-H は、放射線環境が安定した軌道をとる予定なので、諸外国の衛星に比べ、バックグラウンドが安定し、広がった暗いX線放射への感度が高い点も特徴である。ミラーは、SXS と同じものを用いる。

HXI (Hard X-ray Imager) [10] + **HXT (Hard X-ray Telescope)**
(硬X線撮像システム)

HXI は、撮像・分光型の半導体検出器で、いままで撮像することができなかった硬X線の帯域 (5 – 80 keV) のX線画像を撮影することができる。4 層の Double-sided Silicon Strip Detector (DSSD) と 1 層の CdTe パッドを重ねた構造になっている。ここに入射し、シリコン半導体でコンプトン散乱した X 線光子を検出することで、光子のエネルギーと入射方向を決定することができる。本体である半導体検出器を覆うシンチレータとの反同時係数や、多層の DSSD で検出した天体方向外の信号を切り捨てて、効率よくバックグラウンドを除去できる。このため、高感度で硬X線を検出できる。HXI の帯域の

[*1] 宇宙航空研究開発機構 (Japan Aerospace eXploration Agency)
[*2] アメリカ航空宇宙局 (the National Aeronautics and Space Administration)
[*3] 欧州宇宙機関 (European Space Agency)

X線は全反射では集光できないので、X線ミラー (HXT) はブラッグ反射を用いて集光する。焦点距離も 12 メートル必要なため、衛星の伸展部に HXI の検出器を載せる予定になっている。

SGD (Soft Gamma-ray Detector) [11]
(軟ガンマ線分光システム)
SGD は、分光型の半導体検出器である。衛星全体で 2 台設置する予定である。24 層の DSSD と 2 層の CdTe パッドで、入射光子のコンプトン散乱から光子のエネルギーを検出する。HXI と同じバックラウンドの除去方法を用いるだけでなく、検出器上部のコリメータが天体方向以外の光子や荷電粒子を吸収するので、過去最高感度で硬X線から軟ガンマ線の帯域まで (10 – 600 keV) の光子を検出することができる。また、コンプトン散乱の異方性を検出することで、偏光計としての役割も期待されている。SGD をシールドしている BGO シンチレータは、バックグラウンド計数だけでなく、硬X線〜ガンマ線のモニター観測にも用いられ、ガンマ線バーストなどの突発現象を検出することができる。

SXS + SXT-S (軟X線分光システム)	焦点距離	6 m
	有効面積	210 cm^2 (at 6 keV)
	エネルギー帯域	0.3 – 10 keV
	角分解能	<1.7 arcmin
	画素数	6 × 6 pixel
	視野	∼3 × 3 arcmin
	エネルギー分解能	< 7 eV (FWHM, at 7 keV)
	時間分解能	∼80 μsec
	動作温度	47 mK
SXI + SXT-I (軟X線撮像システム)	焦点距離	6 m
	有効面積	360 cm^2 (at 6 keV)
	エネルギー帯域	0.3 – 12 keV
	角分解能	<1.7 arcmin
	画素数	1024 × 1024 pixel
	視野	∼35 × 35 arcmin
	エネルギー分解能	< 150 eV (FWHM, at 6 keV)
	時間分解能	4 sec
	動作温度	-120 ℃
HXI + HXT (硬X線撮像システム)	焦点距離	12 m
	有効面積	300 cm^2
	エネルギー帯域	5 – 80 keV
	角分解能	<1.7 arcmin
	視野	∼9 × 9 arcmin
	エネルギー分解能	1 – 2 keV (FWHM, at 60 keV)
	時間分解能	∼10 μsec
	動作温度	-20 ± 5 ℃
SGD (軟ガンマ線分光システム)	有効面積	>200 cm^2 Photo absorption mode (at 30 keV)
		>30 cm^2 (Compton mode, at 100 keV)
	エネルギー帯域	10 – 600 keV
	視野	0.55 × 0.55 deg^2
		<10 × 10 deg^2
	エネルギー分解能	2 keV (FWHM, at 40 keV)
	時間分解能	∼10 μsec
	動作温度	-20 ℃

表 1.1　ASTRO-H 搭載予定の観測機器の性能 ([6]〜[11]、および SXS, HXI 内部資料をもとに作成)

1.3 時刻性能の目標と要求

図 1.4 は、「すざく」HXD が検出した最も有名なパルサーのひとつ「かにパルサー」の光度曲線 (時間と明るさの関係を示すグラフ) である。かにパルサーの自転周期はおよそ 33 ミリ秒で、その光度曲線は図 1.4 のように 2 つの鋭いピークを持つ構造をしている。このような構造を明らかにするためには、時間分解能の向上が欠かせない。時間分解能とは、各イベントにどれだけ細かく時刻情報を付けるか、を示す値である。例えば、1 周期 33 ミリ秒を 10 ビンでしか表せない程度の時間分解能 (33 msec / 10 ビン = 3.3 msec) しかなかったら、2 つのピーク構造すらはっきりと見えないことになるだろう。

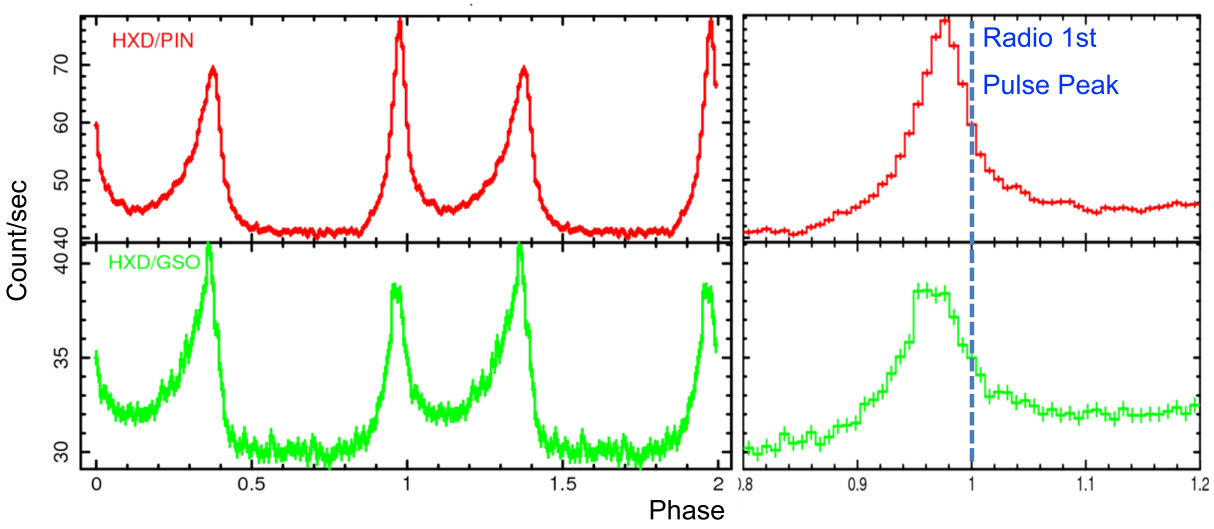

図 1.4 「すざく」衛星搭載 HXD で検出した、かにパルサーの光度曲線。各グラフの縦軸は 1 秒あたりの X 線カウント、横軸はパルス位相 (phase) を表す。phase の間隔の大きさ 1 が、パルスの 1 周期分に相当する。1 周期あたりの時間ビンは 128 ビン。上段 (赤) は PIN 型半導体の 10 – 70 keV、下段 (緑) は、GSO シンチレータの 70 – 300 keV の帯域である。左:2 周期分、右:青が電波で観測されたパルスの到達フェーズ。電波の 1 つ目のパルス・ピークの位置とのずれを拡大したもの

もっとも速く自転する中性子星パルサーは「ミリ秒パルサー」と呼ばれるタイプのものである。現在、最速のもので自転周期がおよそ 2 msec のミリ秒パルサーが見つかっている [5]。ASTRO-H が目指すサイエンスから、この明滅信号を、例えば時間ビン 200 ビンで分解して光度曲線を描くことが要求される。このとき必要な分解能は

$$2 \text{ msec}/200 \text{ ビン} = 10\ \mu\text{sec}$$

となる。本論文では、10 μs を ASTRO-H における時間分解能の目標値とする。

また、絶対時刻とのずれを表す絶対時刻精度は、他の衛星や地上の望遠鏡との同時観測において時間変動を追う時に重要になる。例えば、図 1.4 では、Jodrell Bank Observatory による電波観測から得られた、かにパルサーの自転周期を参照[*4]し、2 周期分の光度曲線を作成している。この時、電波観測で 1 つめのピークの位置を phase$= 0, 1, \cdots$ の位置に合わせている。すなわち、この図から電波と X 線のパルスピークのずれの時間を測ることができる。かにパルサーの場合、可視光・X 線・ガンマ線に対する電波の遅れは 200-300 μs となることが知られている [12]。これよりも 1-2 桁低い精度で遅れを測りたいのであれば、必要な絶対時刻精度は数十〜数 μs となる。本論文では、30 μs を ASTRO-H における時刻精度の目標値とする。ただし、本論文では単に時刻精度と言った場合、絶対時刻精度もしくは時間分解能 (から制限される時刻そのものの精度) のうち大きいほうの値を指すことのする。

[*4] http://www.jb.man.ac.uk/~pulsar/crab.html

1.4　本論文の目的

ASTRO-H における衛星内でのネットワーク通信には、いまや世界標準となった衛星組込ネットワーク「SpaceWire」(後述) が採用される。ASTRO-H は、日本ではじめて、SpaceWire を本格的に採用する大型科学衛星であり、かつ 10 μs 程度の高い時刻性能が必要とされる。衛星に関連するネットワークをすべて標準化する試みは世界でも類をみない。今後 SpaceWire の利用が計画されている世界各国の衛星にとっても、ASTRO-H はいわば「SpaceWire 標準化の実験台」というべき役割を担っている。

本論文では、ASTRO-H 衛星で現在採用されている SpaceWire ネットワークを用いた時刻配信システムに基づき、高い分解能の時刻をつける方法を議論する。さらに、我々が選んだ方法で時刻をつける機能を SpaceWire 搭載機器に実装して、時刻精度を測定する。これらをもとに ASTRO-H 衛星で達成可能な時刻精度を見積もる。

第 2 章

次世代衛星組込ネットワーク SpaceWire

現在、人工衛星や科学衛星の規模が大きくなるにつれて、膨大な開発の手間や資金がかかっている。これは、衛星ごとにデータ収集システムを専用設計で構築している事も一因である。一見して衛星ごとに最適なデータ収集システムを構築することは手間を省くことに繋がりそうであるが、ゼロから設計し開発するコストがかさんだり、機器どうしの整合などに時間がとられてしまうため、膨大な手間とコストがかかることになる。また、これは衛星個々だけに閉じた最適化なので、将来衛星へ資産や知識を継承しにくい。

SpaceWire は、このような衛星搭載のデータ収集システムの問題点を解決するためのネットワーク規格の一つであり、世界の科学衛星で統一的な規格を目指して標準化の議論を進めているネットワークプロトコルである。当初、ESA が IEEE1355 をもとに宇宙機用のプロトコルとして規格化した。現在、ESA のほか、JAXA、NASA、ROSCOSMOS[*1] などが中心となって標準化作業を行っている。衛星内機器の通信インターフェースを統一することで、短期間に開発・実証を行い、人件費や実験のコストをおさえることが大きな目的の一つである。また、SpaceWire を用いたネットワークで構成された衛星であれば資産や知識を継承していくことができる。

前節でも触れたように、ASTRO-H 衛星は、日本の将来の科学衛星のパスファインダーとなるべく、ほぼ初めて、本格的に SpaceWire 規格を採用した衛星となる。本修士論文では、国内外で標準化がすすめられている SpaceWire 規格のうち、時刻付け機能に特化した開発実験がテーマであり、ASTRO-H 衛星の要求精度を満たすかを、実際の SpaceWire 搭載の試験基板を用いて検証することが目的である。この章では、SpaceWire の規格 [13] の概要を解説し、時刻配信に重要な最優先時刻コード「Time-Code」の規定を詳細に述べる。また、SpaceWire の最大の特徴である Remote Memory Access Protocol (RMAP) の規格 [15] を紹介する。

2.1 SpaceWire

衛星開発では、衛星寿命を達成できる高い信頼性、太陽光発電でも駆動する過酷な電力制限、ロケットによる打ち上げから迫られる重量制限、宇宙空間の放射線環境に耐えられること、が求められる。衛星内の統一ネットワーク・プロトコルでは、さまざまな衛星や機器に対応できる柔軟性も求められる。

SpaceWire[13] は、衛星内機器間でデータ通信を行うための高速シリアルリンクの標準規格である。SpaceWire をハードウェアまで統一して使うことで、試験の手間を省きコストを削減することができる。信号伝送には Low Voltage Differencial Signal (LVDS) を採用している。 0.3 V と低い電圧で信号を伝送するので、電力がかからない。また、放射線などの + と − の信号をクロスさせて伝送する差動式を採用しているので、絶対電位でデジタル情報を伝送する場合に比べ、ケーブル全体の絶対電位が振られ

[*1] ロシア連邦宇宙局 (Russian Federal Space Agency)

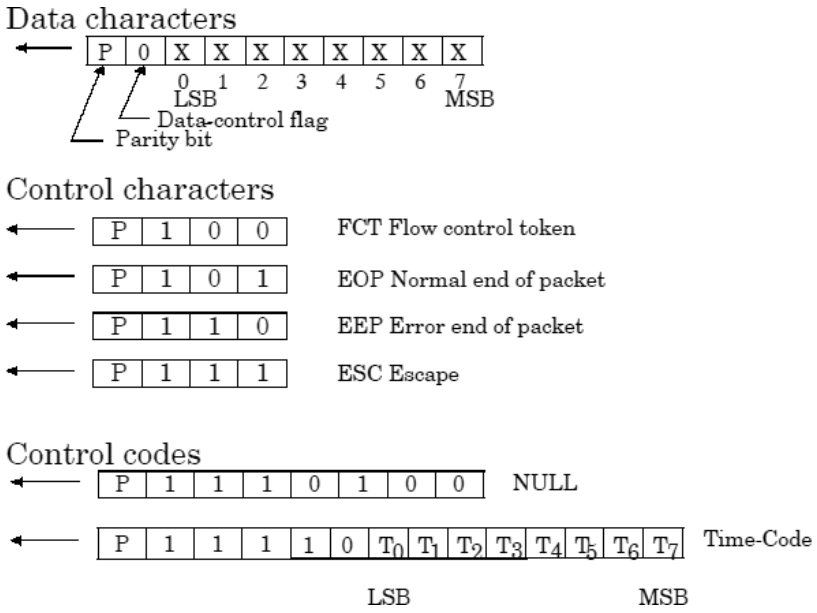

図 2.1　上段:Data Character。中段 4 つ:Control Character。下段 2 つ： Control Codes である NULL と Time-Code の中身 ([13] より)

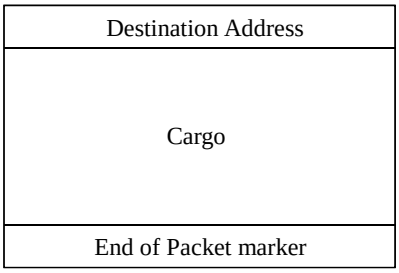

図 2.2　SpaceWire 規定によるパケットの型の定義

るようなノイズに強い。ひじょうにゆるい規定にとどめることで、様々な用途の衛星にあった柔軟な設計ができることが特徴である。 並列同時転送が可能であり、ある回線が切れた場合でも、迂回ルートを使って目的先にデータを届けることができる。このため、さまざまな冗長系を構成できることも大きな特徴の一つである。また、ルーターの機能により、メッシュ状のネットワークを構成することもできる。データ転送速度は、1 ラインあたり 2〜400 Mbps の範囲で可変である。

SpaceWire はプロトコルが簡便なので、容易に FPGA (Field Programmable Gate Array：5.1.1 節, 27 ページ) などのハードウェアに実装できる。

SpaceWire のデータリンク上を流れる情報は、すべてパケットの集合で通信される。それぞれのパケットはすべて図 2.1 の Data Character と Control Chatacter の組み合わせだけで構成される。

Data Character

1 bit のパリティ (parity)、1 bit の Data-control flag のあとに続いて 8 bit のデータ部をもつ。Data-control flag は Data Character を示す 0(ゼロ) である。8 bit のデータ部は Lower Significant Bit (LSB) から送信される。

Control Character

1 bit のパリティ、1 bit の Data-control flag に続いて 2 bit の制御コード部をもつ。Data-control flag は Control Character を示す 1 である。2 bit の制御コード部の値によって、FCT(フロー制御トークン)、EOP(正常パケット終端)、EEP(異常パケット終端)、ESC(エスケープ) の 4 つに分類される。

SpaceWire のパケットは、宛先 (アドレス) を指定して任意のデータを送るパケットと、内容があらかじめ決められている最優先コード (Control Code) の 2 種類に分けられる。前者のパケットの型を図 2.2 に示す。Destination Address は、パケットの送り先を示すアドレス部で、0 またはそれ以上の個数の Data Character で構成される。Cargo は、パケットのデータ部である。End of packet marker は、Control Character の EOP または EEP である。SpaceWire の規定では、パケットの先頭を示すフラグがないため、End of packet marker の次がパケットの先頭だと解釈される。人工衛星では、搭載機器へのコマンド指令や、機器から収集したデータ（テレメトリー）といった「SMCP メッセージ」と呼ばれる情報を 、CCSDS 勧告の Space Packet にまとめて、機器間、地上衛星間のやりとりを行う際に、SpaceWire が用いられる。SpaceWire のレイヤーを細分化してみると、前述したようなパケットの集合体で、疑似的に双方向通信が実現されている。後者の Control Code は図 2.1 に示すように、Data を送信していない状態を示す NULL と時刻コード Time-Code がある。NULL は Control Character の ESC に続くパリティ 0 の FCT の合計 8 bit で構成される。Time-Code は Control Character の ESC に続くひとつの Data Character の合計 10 bit で構成される。Time-Code については本修士論文のテーマである「時刻付け機能」に直結したパケットであるので、次の節で詳しく述べる。

2.2 Time-Code

SpaceWire は、相当に自由なネットワーク設計が可能なのは利点でもあるが、通信の相互タイミングが規程しにくいという欠点もある。そこで、全機器を同期させるタイミングを規程するために考案されたのが「Time-Code」という最優先時刻コードである [14]。時刻コードがすべてのノード (SpaceWire リンクで接続された機器) で統一されるためには、Time-Code のマスターを定義して他のノードと区別するための時刻 ID をつける必要がある。SpaceWire Time-Code では 64 bit すなわち 0〜63 の 2^6 個の ID を振る。また、マスターから Time-Code を発信するタイミングを厳密に規定する Tick 信号が必要である。SpaceWire ではこれを Tick 信号と呼ぶ。以上から、SpaceWire では、64 通りの ID をもつ Time-Code と Tick が考案された。

64 通りの ID を持つ Time-Code も Data Character と Control Code の組み合わせで構成される。詳細は図 2.1 の通りであり、10 bit の Data Character のうち 6 bit が時刻情報部に使われ、これが ID となっている。すなわち、Time-Code は 0 – 63 までの Data をもったものがそれぞれ 2^6 Hz で配信される。Control Flag は、ユーザーが自由に使うことができる 2 つのフラグで、とくに Time-Code 配信のための機能はない。

Time-Code で用いられる Tick は、Time-Code が出たことを示すフラグのようなものである。ノードは、Tick を受信すると Time-Code を受信したと解釈する。そして、現在それぞれのポート (通信を行うためのソフトウェアやハードウェアの末端部分 (インターフェース) のこと) で送信している Character(NULL, FCT, Data character) の送信が完了するとすぐに、受信ポート以外のすべての SpaceWire ポートに Time-Code を送信し、ブロードキャストしてゆく。Time-Code を送信するときは、同時に Tick も送信する。

Time-Code の具体的な配信方法を、図 2.3 に示す。まず、それぞれのノードやルータは 6 bit の時刻カウンタ (Time Counter)、一つ以上の SpaceWire インターフェース (I/F) を持っている。それぞれ I/F は、SpaceWire Link から Tick を受け取るための TICK_IN という input 信号を持っている。(A) ノードやルータが、Time-Code が来たことを示す Tick という信号を受け取ると (すなわち、TICK_IN が 1 になると)、時刻カウンタがインクリメント (1 だけ増加) される。(B)Time-Code の Control Flag は、受け取った Time-Code のものをそのままコピーする。(C)Time-Code の時刻部分の 6 bit は、TIME_OUT に新しい時刻カウンタの値をコピーする。

Time-Code といえど、SpaceWire での通信の際に、パケットがロスする場合がありうる。Time-Code が正しく配信されているかをチェックするために、各ノードで前回の Time-Code の ID を「時刻カウンタ」として保持し、次にきた Time-Code の ID が繰り上がったことを確認し、Time-Code を配信してゆく。ルータの場合、この Tick はすべてのポートに伝播するので、すべてのポートから Time-Code が送信される。図 2.4 に、2 ノード間における Time-Code の配信を示す。点線は SpaceWire リンクを、四角はノード (N1, N2, $\cdots$) またはルータ (R1, R2, $\cdots$) を、四角の中の数字はノードやルータの持つ時刻カウンタの値を示す。(A) は初期状態である。(B)Tick が N1 に入り、N1 の時刻カウンタがインクリメントされる。(C)N1 から Time-Code 1 が送信される。このとき逆戻りはしない。(D)N2 は Time-Code 1 を受信する。これは N2 の時刻カウンタより 1 大きいので、N2 は時刻カウンタをインクリメントし、Time-Code(および Tick) を送信する。

円環状のネットワークの場合を図 2.5 に示す。(A)R1 に Tick が入り、R2 と R3 へ Time-Code が送信される。(B)R2 と R3 は受信した Time-Code が自身の時刻カウンタより 1 大きいことを確認し、Time-Code を送信する。(C)R3 から R2 へ送られた Time-Code は、R2 の時刻カウンタと同じ値である。すなわち、受信した Time-Code が時刻カウンタより 1 大きくない。この場合、受け取られた Time-Code は捨てられ、R2 は Time-Code を出さない。こうすることによって、同じ Time-Code が繰

り返し送信されるのを防ぐことができる。

Time-Code が SpaceWire リンクの経路で失われたとき、そのあとで送られる Time-Code が正しいなら通常の動作に戻るようにするべきである。図 2.6 に、Time-Code が失われた場合の動作を示す。(A)N1 から送信された Time-Code 1 が N1 と N2 の間の SpaceWire リンクで失われたとする。(B)N1 は次の Tick で Time-Code 2 を N2 に送信する。しかし、この Time-Code は N2 の時刻カウンタより 1 大きいわけでも同じでもない。この場合、N2 は受信した Time-Code を時刻カウンタにコピーするが、Time-Code は送信しない。(C) さらに次の Tick で N1 から N2 へ Time-Code 3 を送信する。N2 は、受信した Time-Code が時刻カウンタより 1 大きいので、(D)Time-Code を送信する。これによって通常の動作に戻ることができる。N2 より先にもノードやルータがあれば、同じような連鎖が起きていく。通常の動作に回復するまでに必要な Tick の回数は、ノードやルータの数に依存する。

Tick や Time-Code の時刻を使うと、ネットワーク上のノードやルーターが同期できる。同期のタイミングは、Time-Code の刻みを変えることで、ユーザーの好みに設定できる。ASTRO-H では、Time-Code を用いて同期を行うだけでなく、秒以下の衛星時刻 (後述) の配信に積極的に利用される。

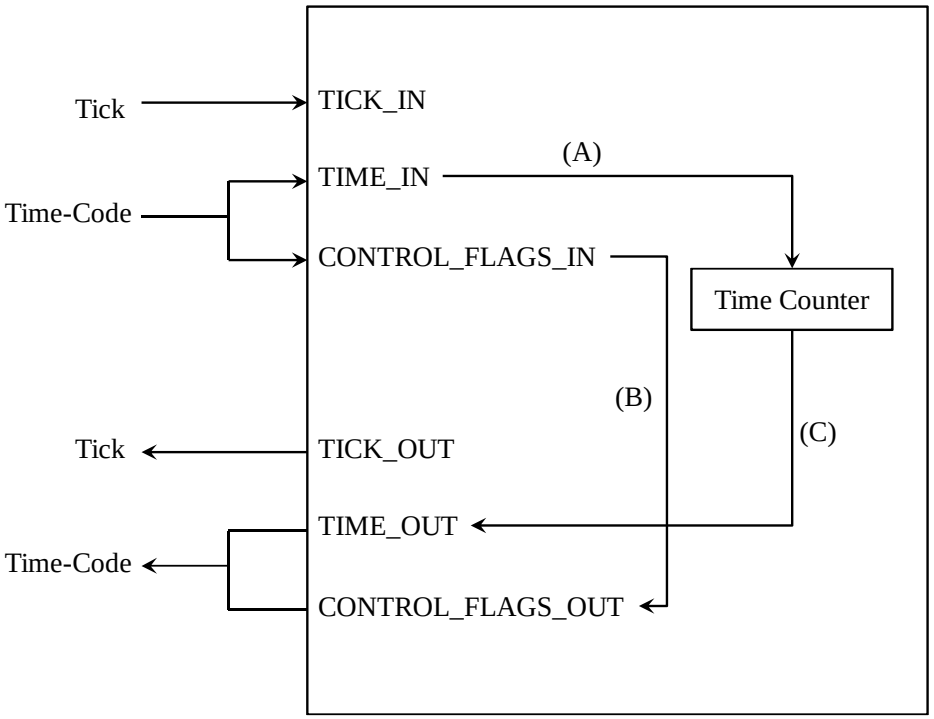

図 2.3 あるノードもしくはルータを例にした、Time-Code の配信の仕組み。詳細な説明は文中を参照

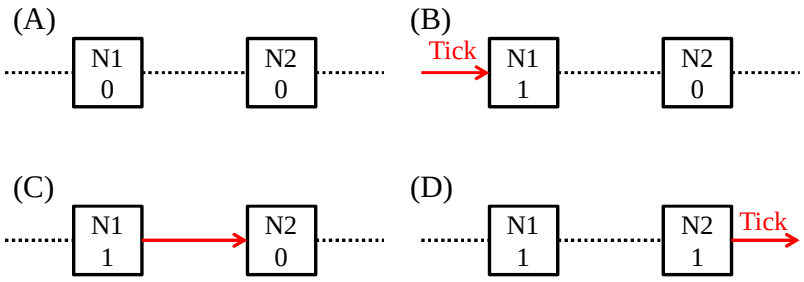

図 2.4　2 ノード間における Time-Code 配信

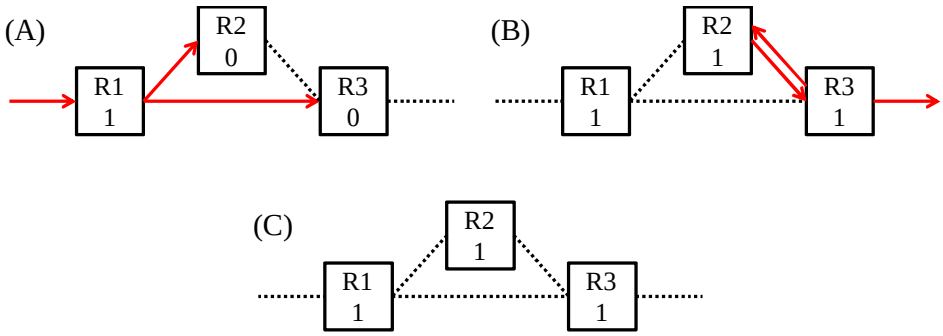

図 2.5　円環状ネットワークにおける Time-Code 配信

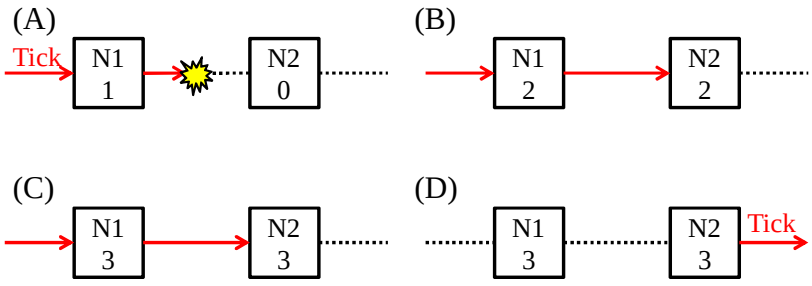

図 2.6　Time-Code が失われた時

2.3 Remote Memory Access Protocol (RMAP)

ネットワークの末端機器のメモリー空間を操作するためには、その末端機器に Central Processing Unit (CPU)[*2]を搭載するのが最も安易な解である。しかし、衛星内のスペース、電力、重量等の制限のなかで末端機器まで CPU を搭載することが難しく、自由な制御体系が組みにくい状況にあった。日本の科学衛星では、PIM という特殊なシステムで CPU 非搭載の機器のメモリー空間まで操作できるシステムがあり成功をおさめていたが、SpaceWire でも同様のプロトコルとして RMAP が提唱されている。Remote Memory Access Protocol (RMAP) [15] は、上位の CPU 搭載ノードから、あたかも自分のメモリ空間の一部として、ネットワーク末端機器の搭載メモリにアクセスできる機能を提供する。しかも、SpaceWire の上位のレイヤーに相当する RMAP はプロトコルレベルで規程されているので、末端機器に CPU を搭載する等の特別な実装を施す必要はない事も特筆すべき点である。

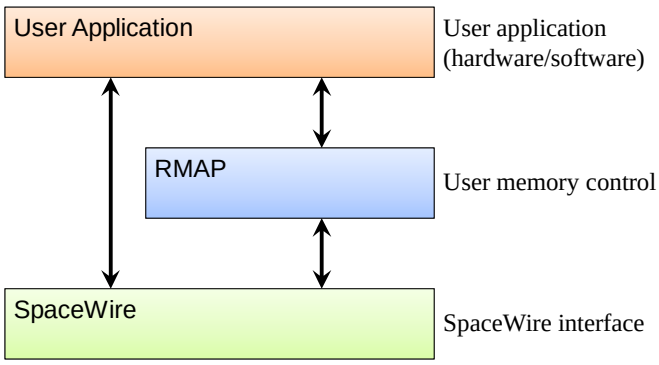

図 2.7 SpaceWire レイヤと RMAP レイヤの関係。ユーザーは SpaceWire ネットワークでのメモリ読出し/書込みに RMAP レイヤを使う。

図 2.8 に簡単な RMAP パケットのやりとりのイメージを示す。RMAP は Command パケットと Reply パケットの組 (Transaction) をひとつの単位として動作する。Command を出す側を Initiator、受け取る側を Target と呼ぶ。RMAP ではよく、write(メモリ書き込み) と read(メモリ読み出し) のふたつの命令を使う。RMAP read の場合、Initiator が Command を送信すると、Target は、要求されたアドレスに対応する値が含まれる Reply パケットを Initiator に返す。Initiator が write Command を送信すると、Target は要求されたアドレスのメモリを書き換えて、書き換えたことを示す Reply パケットを Initiator に返す。

RMAP パケットの送り先を Destination、送り元を Source と呼ぶ。すなわち、Command パケットの Destination は Target、Source は Initiator である。また、Reply パケットの Destination は Initiator、Source は Target である。通常、アドレスやデータ長などを表す部分を RMAP のヘッダと呼び、Destination や Reply パケットを作る際に必要な Source のパス・アドレスなどが書かれている。データ部 (Data) は 8 bit から 16 Mbyte まで可変である。

ASTRO-H では、データ収集をつかさどる搭載コンピュータ SMU(Satellite Management Unit：次章参照) を頂点とする tree 状のネットワーク構成をとり、SpaceWire RMAP を用いて、末端の搭載機器にコマンド[*3]) 指令を送信したり、観測装置などから発生する観測データ (テレメトリー) を収集したりする。テレメトリー・コマンドの送受信の方式としては、RMAP write で送りつける PUSH 方式と、

[*2] コンピュータなどで計算処理を行う部分。アプリケーションから渡された情報を加工し、出力する。中央演算処理装置とも呼ばれる

[*3] 以後、区別のため、RMAP “command” は英字表記、衛星内搭載機器への命令を記したパケットを表す「コマンド」はカタカナ表記とする。

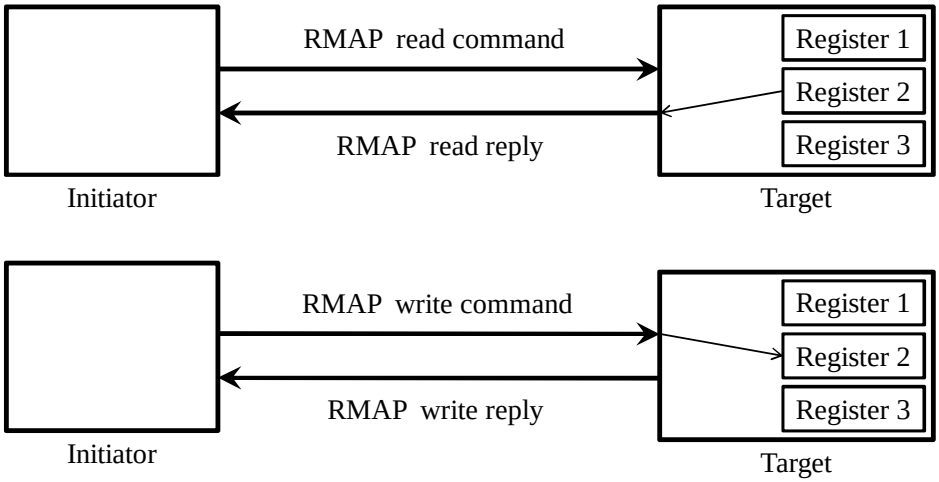

図 2.8　RMAP 通信のイメージ図。上段：RMAP read の場合。下段：RMAP write の場合。

RMAP read で読みに行く PULL 方式とが考えられるが、ASTRO-H では、コマンド送信は SMU から末端機器に PUSH し、テレメトリー受信は SMU が末端機器から PULL する方式をとる。これは、中央コンピュータ SMU の設計を簡略化し、衛星内データのやり取りを SMU で一括管理するためである。特に SMU から各機器へ送る時刻情報は、RMAP write だけでなく、SpaceWire Time-Code でも配信される。次章では、ASTRO-H 衛星内の SpaceWire ネットワークで時刻情報をどのように配信するかを述べる。

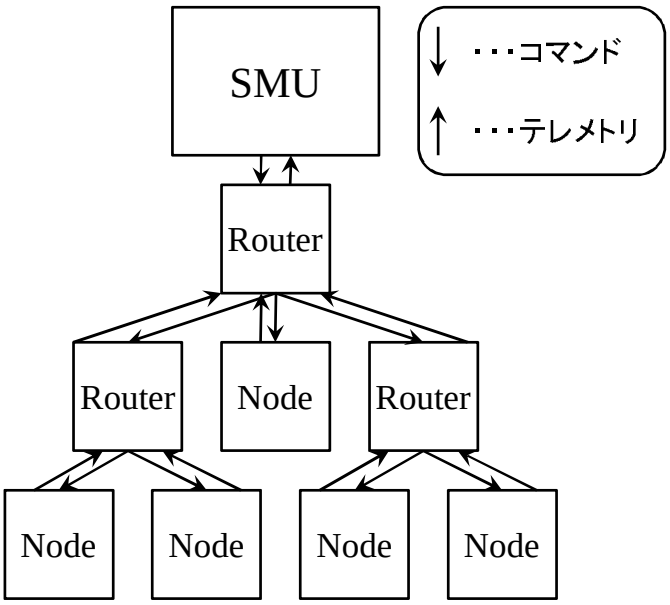

図 2.9　ASTRO-H におけるネットワーク構造のイメージと SMU の位置付。矢印は RMAP write/read command の方向を、N1 や R1 はそれぞれノード、ルーターを示している。SMU 以外のノードやルーターは RMAP command を出すことはできない

第 3 章

ASTRO-H 搭載 時刻配信システム

3.1 衛星内時刻配信の概念

衛星内の検出器で受けたイベントに時刻を付けるためには、衛星内の「時刻配信」が不可欠である。第一章で述べたようなパルサー等の天体解析を行うためには、衛星内の検出器で受けたイベントの一つ一つに「時刻」情報を与えることが重要である。衛星内の各ノードからでる Space Packet に時刻情報を付与する作業を時刻付けと呼ぶ事にする。前述の通り、ASTRO-H のような tree 状のネットワーク構成をとるシステムでは、衛星内の時刻情報をネットワーク最上位にあるノードで一括管理し、それを各ノードに配信する、というやり方を取るのが楽である。ここで管理する「衛星内時刻」を TI と呼ぶ。衛星内ネットワークでは、あるひとつのノードが時刻を生成して配信するという時刻管理を一括して行うべきである。

衛星時刻の精度は、おもに「時刻専用線の有無」と「絶対時刻との同期方法」に依存する。

時刻専用線とは、テレメトリーやコマンドを送るための信号線とは別に、時刻情報だけを送受信するためにある信号線のことである。これにより、時刻情報を送信するとき他のデータが割り込んでタイミングがずれてしまうのを防ぐことができる。たとえば、「すざく」衛星では専用設計のネットワーク構造をもっており、時刻専用線をとおして常時 秒以上の TI の値と、秒以下の正確なタイミングを末端の機器まで送ることができる[*1] [4]。ASTRO-H 衛星の場合、時刻を含むすべての情報は SpaceWire の信号線でやり取りされ、時刻専用線をもたない。このため、衛星時刻が末端の機器に通知されるタイミングがの規定が難しくなる。

また、時刻管理をするノードで絶対時刻と TI を同期させることで、絶対時刻精度が向上する。「すざく」の場合は、各 10 分、一日 5 回、衛星と地上運用局が交信できる際に、中に地上のセシウム時計と TI を照合していた [4]。この方法では、地上のアンテナと交信できない時間帯の時刻は、時刻照合時の値の内挿で概算するしかなく、絶対時刻精度が低下してしまう。静止衛星でない限り、観測時間の大半は地上のアンテナと交信できず、絶対時刻精度が高い時間帯はわずかである。そこで、近年の衛星によくつかわれるのが GPS システムで配信されるの時刻情報である。ASTRO-H では、GPS 衛星の絶対時刻のタイミングから衛星時刻を生成して配信する。衛星軌道上で GPS 衛星を捕捉しているときならいつでも、高い絶対時刻精度を保つことができるのが利点である。

[*1] 正確には、32 bit の衛星時刻 (TI) のうち、秒以上の 24 bit は、時刻専用線で搭載機器の CPU 搭載部のレジスタに書き込まれ、秒以下は 1 Hz の正秒を示すクロックと、512 kHz のクロックとで伝達されている。

3.2　ASTRO-H における時刻配信

3.2.1　衛星のネットワーク構成

ASTRO-H で現在計画されている衛星全体のネットワーク構成 [16] を図 3.1 に示す。まず、おおまかにデータ処理系と姿勢制御系のふたつのネットワークで構成される。データ処理系は、観測データなどを処理するための機器がある。主に、SMU(Satellite Management Unit) や観測機器系のネットワークが含まれる。姿勢制御系は、衛星の向きと光軸中心に対する回転 (衛星の姿勢) を制御する機器から成る。衛星の姿勢制御は、衛星の生死を決めることがあるので、データ処理系から姿勢制御系を制御できないようにする。具体的には、データ処理系と姿勢制御系は別の SpaceWire ネットワークに分けて、AOCP(Attitude and Orbit Control Processor) というノードを通してブリッジ接続させる。

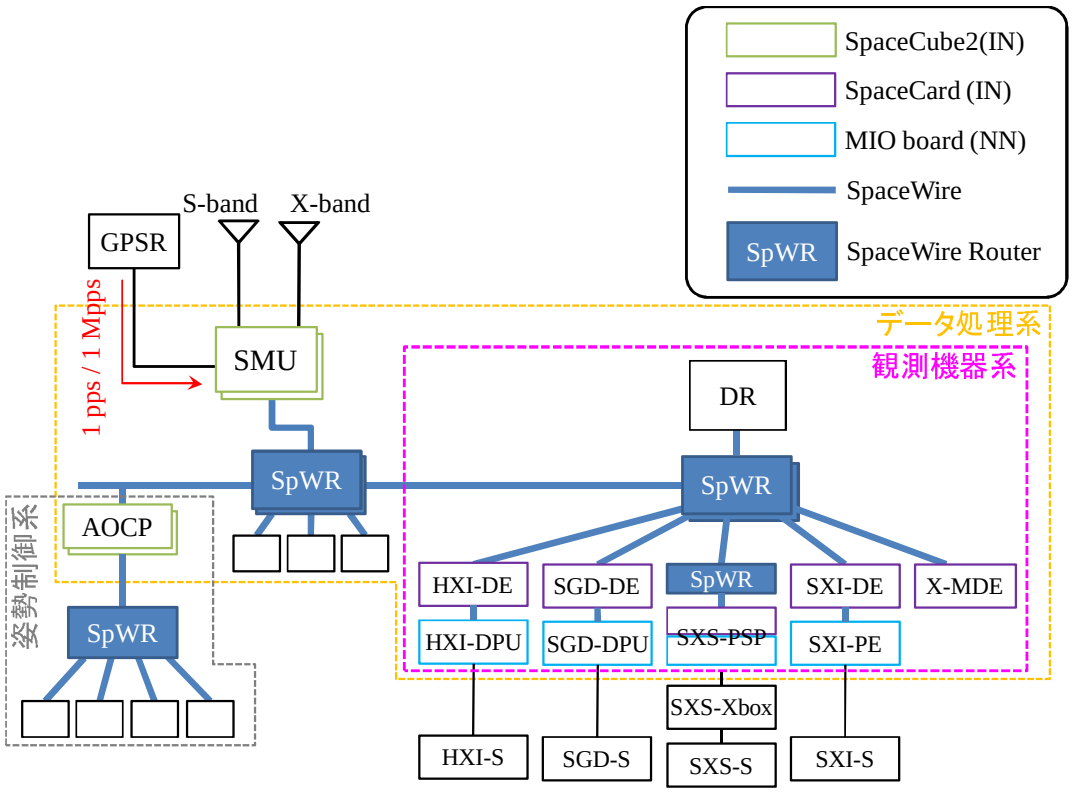

図 3.1　ASTRO-H 衛星のネットワーク構成。上にいくほどネットワークで上位のノードである。凡例の IN はインテリジェント・ノードを、NN はノンインテリジェント・ノードを示す。([16] および ASTRO-H の内部資料をもとに作成)

ASTRO-H 衛星のネットワーク全体を管理するノードは、最上位ノードの SMU である。SMU は、SpaceWire リンクで接続された下位の各ノードへ RMAP write でコマンドを送ったり、各ノードから RMAP read でデータを収集する。また、自身に接続されている GPS 受信機 (GPSR) のクロック[*2] から TI を生成して配信する。さらに、地上交信用のアンテナ (S 帯と X 帯) と接続している。

観測機器系は、DR (Data Recorder) と、SXS, SXI, HXI, SGD の各検出器においてデジタル処理を担う ノードで構成される。後者を CPU をもつノード (インテリジェント・ノード) と、CPU をもたないノード (ノンインテリジェント・ノード) に分ける [17] と、それぞれの役割と名称をそれぞれ図 3.2 のようになる。観測機器系のノンインテリジェント・ノードは、検出器本体 (Sensor : S) から出てくるイベント信号に対して、本論文のテーマである時刻付けや、波高値解析、画像処理、グレード判定、など

[*2] ある一定の周波数の矩形波。「時計」の刻みを与えるために使うのでクロックと呼ばれる。

のデジタルデータ処理を行う FPGA 基板である。これを、上位のインテリジェント・ノードが RMAP read で読出し、不要なデータを除去してから Space Packet (宇宙機用のパケット) にする。インテリジェントノードから送られる情報の中身は、日本が標準化を進めている SMCP [18] の枠組みに沿って生成され、この SMCP メッセージを SpacePacket としてまとめる際には、CCSDS 勧告にそった形式 (CCSDS 形式:[19]) をとる。

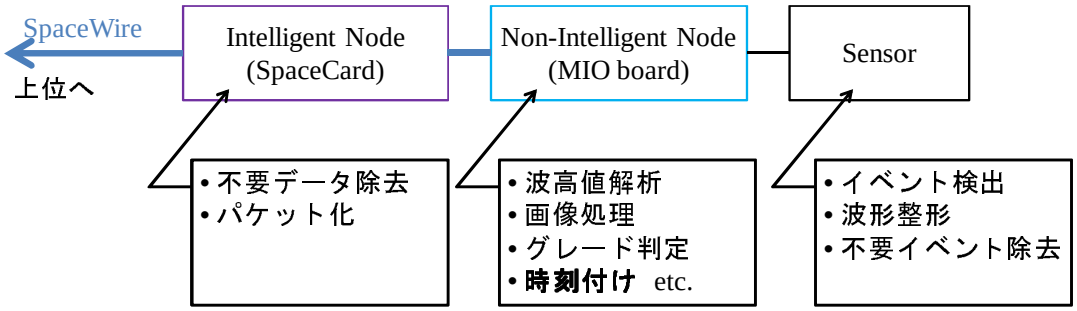

図 3.2 各機器における処理の概要。凡例は図 3.1 と同様。

これにより、機器同士の整合にとられる手間などを省くことができる。SMU には A 社製の SpaceCube2 が採用されている。観測機器系のインテリジェント・ノードには、SXS-PSP の CPU 部、SXI-DE、HXI-DE、SGD-DE が相当するが、B 社製の SpaceCard を用いる予定である。また、観測機器系のノンインテリジェント・ノードには、SXS-PSP の FPGA 部、SXI-PE、HXI-DPU、SGD-DPU が相当し、同じく B 社製の Mission I/O board(以下 MIO board) を使用する予定である。

3.2.2 衛星時刻 (TI) の配信

ASTRO-H で用いる TI の構造は、図 3.3 に示すように、秒以上の TIME DATA 32 bit と秒以下の Time-Code 6 bit の全体で 38 bit である。TI のカバーする時刻の範囲は $2^{32} - 1/2^{6}$ 秒すなわち約 136 年 – 15.6 ミリ秒である。

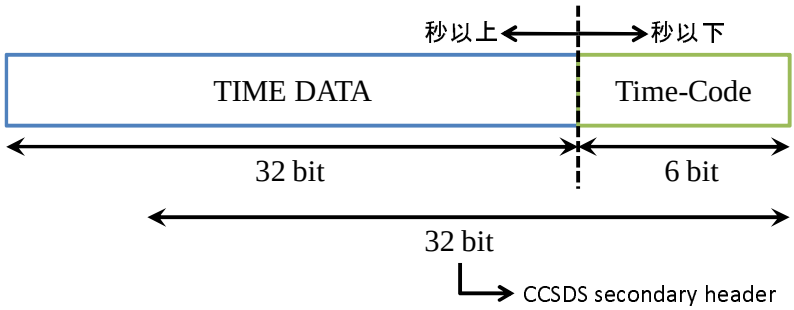

図 3.3 TI のフォーマット

	長さ	時刻範囲	配信方法
TIME DATA	32 bit	$2^{32} - 2^{0}$ 秒 (136 年 – 1 秒)	RMAP write
Time-Code	6 bit	$2^{0} - 2^{-6}$ 秒 (1 秒 – 15.6 ミリ秒)	SpaceWire Time-Code
TI	38 bit	$2^{32} - 2^{-6}$ 秒 (136 年 – 15.6 ミリ秒)	TIME DATA + Time-Code

表 3.1 TI の時刻範囲と配信方法のまとめ

ASTRO-H のネットワークでは、Time-Code を管理するノードを Time Master、時刻を受ける下位のノードを User Node と呼ぶことにする。

Time Master

Time Master は、SpaceWire Time-Code を生成・配信する機能や、その機能をもつノードのことである。ひとつのネットワーク内に同時にひとつのノードが Master 機能を保有できる。データ処理系 (AOCP 含む) のネットワークでは、SMU が Time Master である[*3]。

User Node

User Node とは、Time Master から SpaceWire リンクで配信された Time-Code や衛星時刻を受け取るノードやルーターのことである。ASTRO-H では、SMU を除く全てのデータ処理系のノード (AOCP 含む) は SMU の User Node にあたる。イベント時刻付けの点で特に重要な User Node は観測機器系の MIO board である。

ASTRO-H で採用する予定の GPS 受信機 (GPSR) は、GPS 時刻と同期した 1 pps(正秒のタイミングを与える信号) と 1 Mpps($1/10^6$ 秒周期のクロック) を送信する [20]。SMU は、GPS 1 Mpps クロックから 64 Hz でカウントアップする Time-Code を生成する。Time-Code は 6 bit なので、1 秒でひとまわりする。生成された Time-Code は SpaceWire リンクを通じて下位の User Node へ配信され、末端機器の User Node へと伝播していく。また、SMU は同時に GPS クロックから TIME DATA も生成し、RMAP write によって各 User Node の衛星時刻用のレジスタに書き込む。User Node は、RMAP write で書き込まれた TIME DATA と SpaceWire で配信される Time-Code から、TI を再合成する。

ASTRO-H 衛星では、GPS 故障時に備え、「すざく」方式の時刻付け機能も併用する。すなわち、TI の値だけが含まれた時刻専用の Space Packet を用いて、地上の運用局にあるセシウム時計と TI を照合するルートも確保している。

実際に運用者やデータ解析者が配信された TI を利用するためには、テレメトリとしてノードから出力する必要がある。インテリジェント・ノード[*4]から出力される CCSDS 形式のパケットのセカンダリ・ヘッダに、そのパケットを生成したノードの TI のうち下位 32 bit (TIME DATA 下位 26 bit + Time-Code 6 bit) が埋め込まれる [21]。

3.3 時刻目標達成への課題

以上が、ASTRO-H 衛星に搭載が決まっている設計の概要である。前述のとおり、SMU が衛星全体に SpaceWire を通じて配信する TI の時間分解能は 15.6 ミリ秒である。衛星のバス系としてはこれで十分であるが、ASTRO-H 搭載の天体観測装置の時刻精度としては、1.3 節で述べたような時刻性能要求 (時間分解能 ∼10 マイクロ秒) を達成できない。本修士論文では、科学データに要求された時刻性能を満たす設計を模索し、イベント時刻付けの概念設計を次章で行うと共に、この設計案にもとづいた検証実験も行う。

[*3] 姿勢制御系の AOCP 以外のノードには SMU から送られる Time-Code が到達しない。そのため、姿勢制御系は別のネットワークとして扱われる。本論文では特に姿勢制御系の時刻配信については議論しない。

[*4] SMU も含まれていることに注意する。

第 4 章

高時刻精度を持つイベント時刻付けシステムの概念設計

ASTRO-H 衛星の観測機器に必要な時間分解能と時刻精度は、それぞれ 10 μsec と 30 μsec である (第 1 章)。しかし、前章で述べたように、ASTRO-H 衛星の衛星時刻 (TI) の時間分解能および時刻精度は Time-Code の刻みと同じ 15.6 msec しかなく、観測機器での X 線検出時刻に用いるには不十分な精度である。観測機器系の側で TI とは別により細かい時刻精度をもたせる方法を考える。このように、観測機器で検出した X 線光子一つひとつに時刻情報を与えることを、以下、「時刻付け」と称し、要求精度の時刻付け手法を確立することが、本修士論文のテーマである。

4.1 高分解能時刻を達成する方法

観測機器で検出したX線イベントに高い時間分解能と時刻精度をもった時刻をつけるためには、観測機器系の User Node (実際は MIO board) に要求分解能をもった時刻カウンタをもつ必要がある。本論文では、この時刻カウンタのことを「LocalTime」と呼ぶことにする。

LocalTime をつくる方法は、TI と同期させる方法と、同期させない方法のふたつがある。前者は、Phase Locked Loop (PLL) という方法で、4.1.1 節で詳しく述べる。後者 (4.1.2 節) は、「LocalTime が A という値のとき、TI は B という値だった」という照合を何らかの方法で行わなければならない。ASTRO-H に適切な手法を選ぶには、宇宙機用ハードウェア・ロジックに実装できるか、要求時刻精度を達成できるか、といった点も評価しなければならない。以下に、両者の手法の原理をまとめ、こうした評価を下すべく思考実験を行う。

4.1.1 PLL による衛星時刻クロックへの同期を用いる方式

Phase Locked Loop (PLL) は、クロックの同期に用いられる回路である。はじめ、フランスの H. de Bellescize が 1932 年に Synchrodyne(同期受信機) という名前で提唱した。現在では、テレビ、ラジオ、コンピュータ、携帯電話など、身近な電子機器には欠かせない技術となっている。

PLL の基本的な回路は、図 4.1 のように、以下の 3 つの要素から成り立っている。

位相比較器

2 つの入力信号の位相差を出力する。位相比較器には、デジタル方式とアナログ方式がある。

ループ・フィルタ

ローパス・フィルタ (積分回路) である。位相比較器からのリップル[*1]を含んだ直流信号を平均化し、交流成分の少ない直流信号に変換する。ループ・フィルタには、リップルを取り除くほかに、PLL のループ制御を行うために伝達特性を決めるという役割もある。つまり、ループ・フィルタの時定数を間違えると、位相がロックしなくなってしまう。

Voltage Controlled Oscillator (VCO)
入力の直流信号の電圧によって発振周波数が制御できる、可変周波数発振器である。

PLL 回路は、出力信号を制御して、位相比較器に入るふたつの入力信号の位相が同じになるようにする。すなわち、水晶発振器のクロック周波数 f_{in} と、VCO の出力周波数 f_{out} が同じになる ($f_{in} = f_{out}$)。もし、f_{in} と f_{out} が違う時には、図 4.2 のように、位相比較器から位相差を示す信号 (A) が出る。結果、ループ・フィルタからの出力信号 (B) が変動するが、徐々にある一定の電圧に落ち着き、$f_{in} = f_{out}$ となる。この状態を「ロック」と呼ぶ。

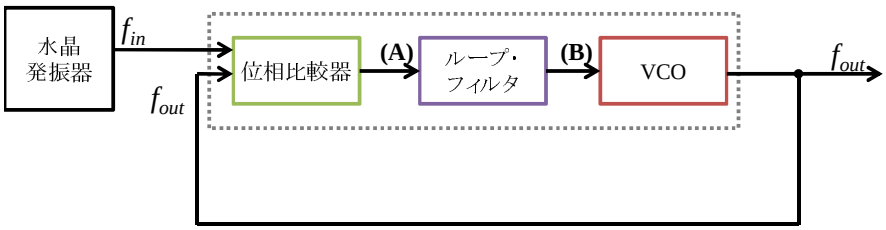

図 4.1　基本的な PLL 回路。灰色点線で囲んだ部分が PLL である

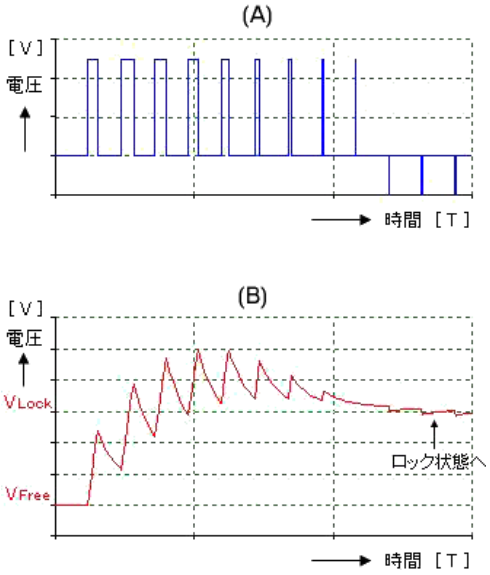

図 4.2　基本的な PLL 回路 (図 4.1) の応答のようす。(A), (B) は図 4.1 に対応している。
(http://gate.ruru.ne.jp/rfdn/TechNote/BasePllTech.asp から引用)

f_{in} より細かい時刻を作り出すためには、図 4.3 のように、VCO から位相比較器へと入力される線の間に分周器を挟む。すると、$f_{in} = f_{out}/N$ となるように制御されるため、VCO からの出力周波数は、

$$f_{out} = N \times f_{in}$$

となる。これで N 倍の周波数のクロックを得ることができる。

[*1] 交流を直流に変換したときに残る電圧振動。

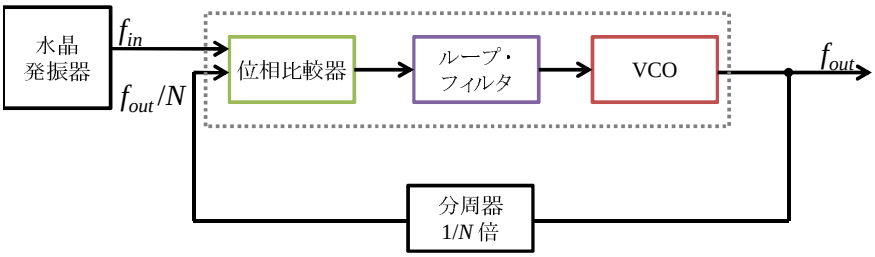

図 4.3 分周器を入れた PLL 回路。灰色点線で囲んだ部分が PLL である

ASTRO-H で Time-Code からより細かい時刻を得るために PLL を使うとするならば、観測系の FPGA 搭載部 (MIO board : 第 3 章) の FPGA にデジタル的に実装する必要がある。この場合、全てのロジックをデジタルで処理するために、前述の各モジュールをデジタル位相比較器やデジタル VCO(電圧ではなく数値で周波数を変える) で代用することになる。雑音を除去できるうえ、動作検証が容易になるが、以下の欠点がある。

1. アナログと比べてロジックを多く必要とする
2. VCO の周波数が低く、ジッタ (時刻不定性) が大きい
3. ロックするまでの時間が長い

特に位相比較器と VCO は、デジタル的に実現するためにはかなりの実装ロジック数が必要になる。上記 2 と 3 の問題を改善するためには、さらに高度な回路を実装しなくてはならない。すでに波高値解析や画像処理など別の機能にロジックを占有されている MIO board に高性能なデジタル PLL を実装するのは非現実的である。さらに、

4. 衛星に要求されるレベルの高信頼性ロジックの入手が困難

という問題もある。デジタル PLL 自体は各所で開発され、フリーな IP core として配布されているものが多いが、実績があり信頼性の高いロジックは、相当なロジック数が必要となる。ASTRO-H 衛星に限らずとも、色々な規模の衛星で SpaceWire ネットワークを有効に用いるためには、末端ノードにばらまく通信素子は、できるだけ小規模なロジックのものに収めたいものだが、衛星搭載用 PLL 回路は (末端機器にばらまくにしては) 高機能すぎる。

そこで、次節では、これらの問題を解決し、現在稼働中の「すざく」HXD でも実績のある、フリーランクロックによる内挿方式を提案する。

4.1.2 非同期クロックを用いた時刻 tick 内挿法

観測機器系の User Node で Time-Code より高い時刻精度をもつ時刻をつくる方法として、Time-Code と同期していない LocalTime をもつという手法がある。LocalTime 自体が TI よりも高い時間分解能を持つクロックであることは PLL 法と同じであるが、LocalTime と TI の同期をあきらめ、定期的に両者を（高い時刻精度で）「照合」することで、ユーザーノードにおいて疑似的に高い時間分解能をもたせるのである。具体的には、図 4.4 のように、MIO board で自身の持つ LocalTime と SMU から受けた TI とを、「照合」することになる。ただし、この手法では、LocalTime が TI と照合されるタイミングが PLL 法に比べてまばらであるため、「照合と照合の間の LocalTime カウンターの進みは一定である」という仮定を置いて、TI よりも高い時刻精度の情報を補完する。本論文では、この手法を 非同期クロックを用いた時刻 tick 内挿法、または簡単に 時刻 tick 内挿法と呼ぶことにする。

時刻 tick 内挿法のためには、(A) X線イベントの検出時刻を高い時刻精度でつけた LocalTime、

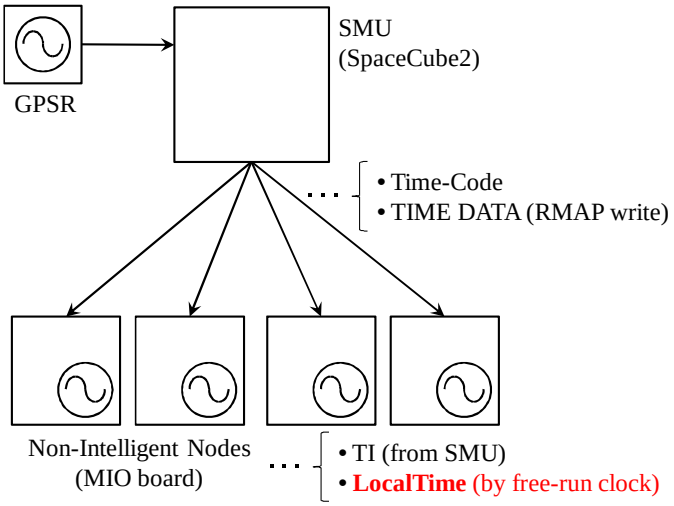

図 4.4　非同期クロックを用いた時刻 tick 内挿法での Time Master と User Node の動作概要。SMU と MIO board の間のネットワークは省略している。

(B) TI と LocalTime の照合データ、のふたつをテレメトリに出す。そのパケットのイメージを図 4.6 に示す。(A) は、X線イベントごとに LocalTime をラッチし、波高値、検出ピクセル位置などのイベント情報を含む Space Packet (Science Data Packet と呼ぶ) にして、SMU に RMAP read されることにする。その際、SpacePacket が生成された時点の TI がヘッダーに含まれるので、(A) の光子検出時点付近の TI 値もテレメトリーにでる。(B) のほうは、別の Space Packet (時刻の HK packet) として SMU に RMAP read される。

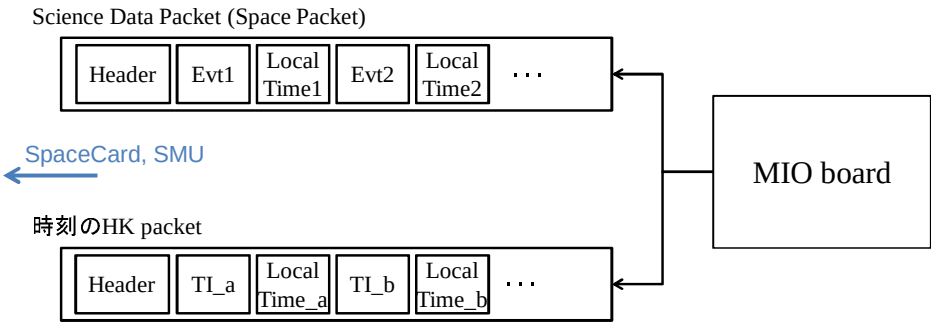

図 4.5　時刻 tick 内挿法における MIO board からのテレメトリ出力。Evt1, Evt2, ... には、波高値、検出ピクセル位置などの情報が含まれている

LocalTime をイベントに付けただけでは、GPS 時刻と同期した時刻になっていない。これを改善する方法を図 4.6 に示した。MIO board では、ある一定のタイミングで TI と LocalTime の照合表を記録しておき、テレメトリとして出力し地上へと送る (この照合表をテレメトリ出力する際のパケットを、本論文では「時刻の HK パケット」と呼ぶ)。地上で、衛星管理者や解析ユーザーなどが、LocalTime と TI の相関データをこの照合表から調べ、各データ点間をある関数で内挿する。そして、イベントについている LocalTime の値をその関数に代入すれば、そのイベントの TI が推定でき、その分解能は LocalTime と同じになる。

ただし、イベント一つ一つに付く LocalTime のデータ長はあまり長いものにはできないため、137 年持つ TI よりもずっと短い時間で桁上がりが生じる。この方法の時刻付けに関しては、観測機器系のデジタル処理部 (具体的には CPU をもつ SpaceCard) に、「LocalTime が桁上がるまえに TI の値をイベントの SpacePacket に編集する」ことが要請される。この条件を守ることで、イベントの TI から、どのラウンドの LocalTime であるかが推定でき、「時刻 HK」から正確な TI を 1:1 で対応づけることで、

より長い時刻情報も付加できるようになる。

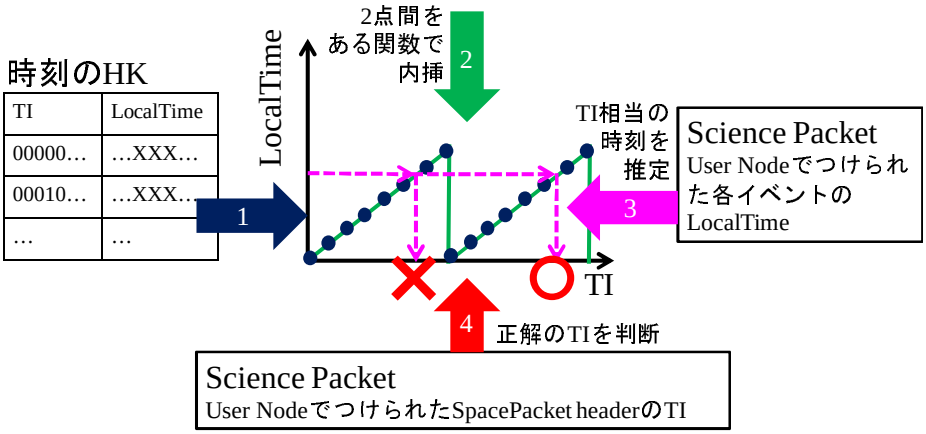

図 4.6　具体的な 時刻 tick 内挿法で高い時刻精度を得る方法。1. 時刻の HK (TI と LocalTime の照合表) から、TI と LocalTime の相関をとる。2. データ点間をある関数で内挿する。3. 各イベントについている LocalTime を 2 の関数に代入して、TI を推定する。LocalTime のカウンタが回りきるので、推定される TI は複数ある。4. SpacePacket のセカンダリ・ヘッダに埋め込まれた TI から、正解の TI を選定する。

この方法では、衛星軌道上で TI とより細かい時刻 (LocalTime) を同期させる必要がないため、単純なロジックで済む。よって、デジタル PLL で問題となっていたロジック数の多さを改善できる。また、61 μsec の時間分解能を誇る「すざく」衛星でも実績のある方法である [4]。

4.2　非同期クロックを用いた時刻 tick 内挿法における時刻性能

以上から、ASTRO-H 衛星に搭載するハードウェア・ロジックに実装できるかという観点からは、フリーランクロックを用いた時刻照合による方式が、もっとも適していると言える。次に、ASTRO-H に科学面から要求されている時刻精度を達成できるか検証する必要がある。この節ではフリーランクロック方式で検証すべき、時刻精度を悪化させる原因となる現象をまとめる。

4.2.1　Time-Code のジッタ

ASTRO-H 衛星の場合、時刻専用線を使わないため、SpaceWire の Time-Code で機器間の同期が行われるが、ユーザーノードに到達する時刻がゆらぐ。一般に、信号の時間的なずれや揺らぎをジッタといい、おもに信号の伝送経路の影響で発生する。Time-Code ジッタのおもな原因は、SpaceWire を通じて配信される Time-Code が、他のデータやコマンドに割り込むタイミングがランダムになることである。すなわち、ここで見積もるジッターは、PLL 法、非同期クロック法のいずれでも時刻精度悪化の原因になる。この節では、その割り込み課程を具体的に述べ、ジッタの値を見積もる。

SpaceWire では優先度が設定されていない。しかし、Time-Code はその例外で、最優先時刻コードである。つまり、Time-Code は、RMAP パケットより小さい単位の Character の間に割り込むことができる (この Character は 2 章で説明した Data Character や NULL, FCT である)。Time-Code のジッタが発生する過程を図 4.7 に示す。SpaceWire リンクでいま送信完了した他のデータ (Data Character) 直後に Time-Code (Tick) が送信されると、Time-Code の遅延は固定遅延 (ケーブルや他の信号線を通るのにかかる時間) のみとなる。一方、他のデータが送信されているときに Time-Code が送信されると、Tick が出力されてから Time-Code が出力されるまで時間差ができてしまい、固定遅延に上乗せして遅延が発生する。本論文ではこの時間差をジッタと呼ぶ。固定遅延は固定長でゆらがないと予想され

るので、打ち上げ前の衛星などで計測しておけば済むものである。しかし、Tick がどのタイミングで入力されるかによりジッタの値がランダムに変化してしまうので、ジッタの値は時刻の「誤差」となり時刻精度を悪化させる要因となる。

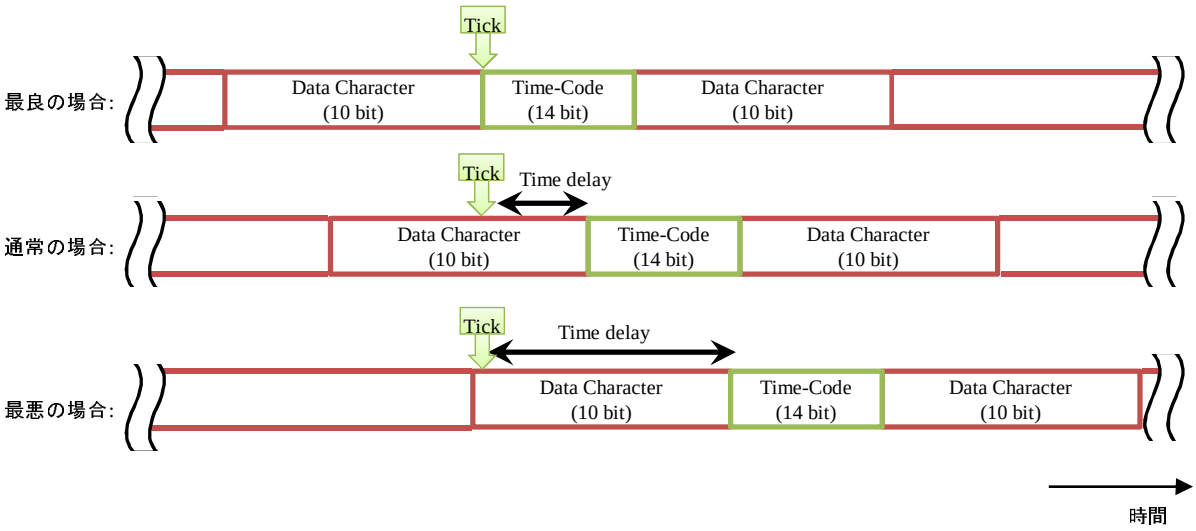

図 4.7　Time-Code のジッタが発生する原因

1 bit 転送するのにかかる時間を 1 clock cycle と言うことにすると、Data Character(10 bit) 転送中に発生する Time-Code のジッタは 10 clock cycle になる。なぜなら、Tick が、Data Character の 0 bit から 10 bit までのランダムな位置で出るからである。リンクレートが R Hz なら、1 clock cycle = 1 / R sec だから、

$$10 \text{ bit} \times 1/R = 10/R \text{ sec}$$

となる。第 6 章で実測結果を述べる 100 MHz リンクにおける NULL(8 bit) 転送中の Time-Code のジッタは、

$$8 \text{ bit} \times 1/(100 \times 10^6 \text{ Hz}) = 80 \text{ nsec} \tag{4.1}$$

になるはずである。ASTRO-H の場合は 20 MHz リンクなので、同様の場合、

$$8 \text{ bit} \times 1/(20 \times 10^6 \text{ Hz}) = 400 \text{ nsec} \tag{4.2}$$

となる。

さらに、ルーターを 1 個, 2 個,$\cdots$ と介すると、このような現象が多重発生することになり、ジッタは 2 倍, 3 倍,$\cdots$ となる。Time-Code の伝送経路中に挟むルーターの個数を N、ルーターを挟まない時の Time-Code のジッタを $\Delta t_{\rm int}$ とすると、$\Delta t_{\rm int} \times (N+1)$ が N 個ルーターを挟んだ時に生じる Time-Code のジッタとなる。ASTRO-H 衛星では、SMU から User Node (MIO board とする) までの経路で $N = 2$ – 3 個である。SpaceCube2, SpaceCard, MIO board それぞれが内部ルーターを持っていることも考えると、SMU から MIO board までの $N = 5$ – 7 個になる。ゆえに 4.2 式から、SMU で送出した Time-Code が MIO board で、

$$400 \text{ nsec} \times (5\text{–}7) + 1 = 2.4\text{–}3.2 \ \mu\text{sec} \tag{4.3}$$

となる。

NULL 通信だけを行っている時、ルーターを N 個に挟んだ場合の Time-Code のジッタの確率分布は、8 つの目をもつサイコロを $(N+1)$ 個同時に振ったときの目の和の確率分布と同じ原理で説明でき

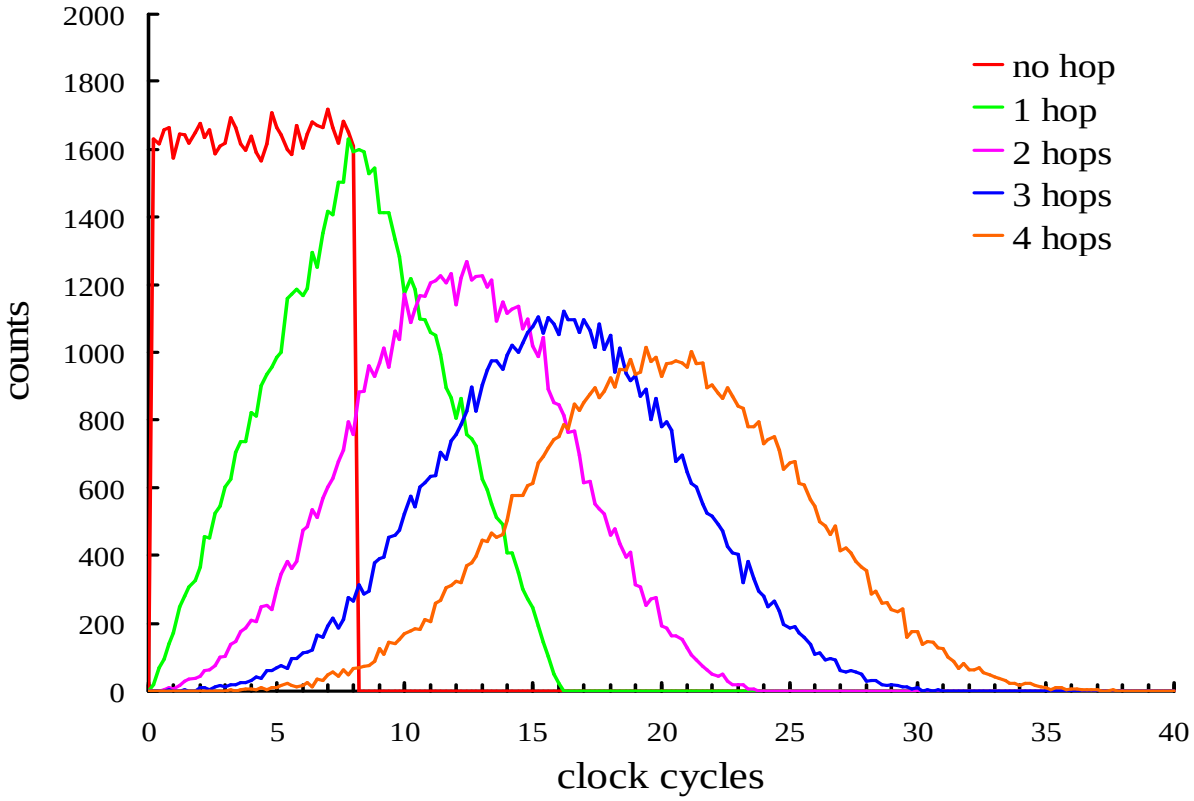

図 4.8 NULL 通信時の Time-Code ジッタの確率分布シミュレーション。赤：ルーターなし、緑：ルーター 1 個、マゼンタ：ルーター 2 個、青：ルーター 3 個、オレンジ：ルーター 4 個。

る。各ルーターにおける Time-Code のジッタ (サイコロなら目の数) を乱数で与え、N 回足し算することでシミュレーションすると、図 4.8 のようになる。

Time-Code のジッタは MIO board で合成される TI の精度にもろに影響する。したがって LocalTime の内挿方式によって推定される時刻精度を悪化させる。6 章 6.1 節では、Time-Code のジッタを実際に SpaceWire 搭載エレクトロニクスを用いて測定し、ASTRO-H 衛星における時刻精度を見積もる。

4.2.2 LocalTime の時刻安定度

もうひとつ、時刻精度に影響するパラメータとして、フリーランクロックの時刻安定度と時刻の HK のサンプル周波数がある。LocalTime のベースとなるフリーランクロックは MIO board の水晶発振器である。水晶発振器は、温度変化やそれ以外の要素によって出力するクロック信号の刻みが変化する。特に、温度が変わると通常の水晶発振器の場合でおよそ 10% も周波数が変わることがある。その特性は発振器によって異なる。また、周波数安定度を上げた「温度補償付水晶発振器」や「恒温槽付水晶発振器」もある。

フリーランクロックを用いた時刻照合で細かい刻みの時刻をつけるとき、時刻の HK のサンプル周波数が長いと、水晶発振器の安定性を拾ってしまうことになる。逆に、サンプル周波数が十分短いと、水晶発振器の安定性をほとんど無視できることになるが、時刻の HK パケットの量が多くなり、他の観測データなどを圧迫しかねない。TI との時刻照合の時間間隔を S sec、LocalTime の安定度を A sec/sec、LocalTime に要求されている時刻精度を Y sec とすると、

$$Y > S \times A$$

となる。すなわち、

$$A < Y/S$$

が要求される。6 章 6.2 節では時刻安定度を実測するとともに、6.3 節でサンプル周波数と時刻精度の関係を SpaceWire 搭載エレクトロニクスで実測する。これらの実験結果から、要求時刻性能を満たす適切なサンプル周波数を見積もる。

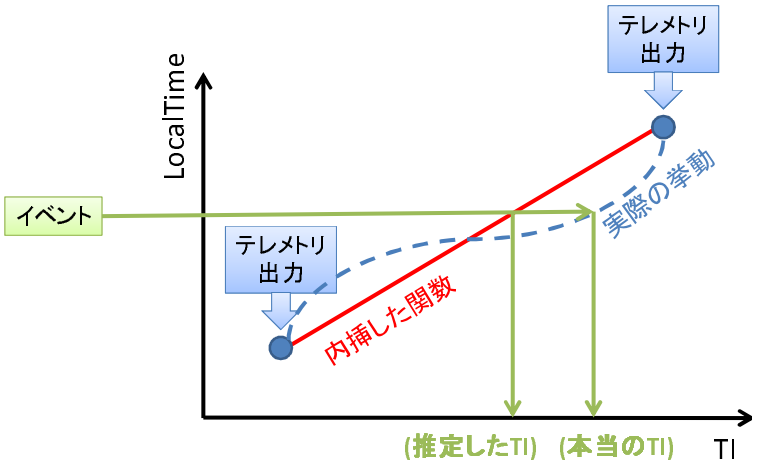

図 4.9　LocalTime の安定度と時刻精度。サンプル周波数が長すぎると水晶発振器の周波数不定性から、内挿した関数が実際の挙動と違ってしまうので、時刻照合による時刻精度が悪化する

第 5 章

SpaceWire 搭載機器への実装

現在 ASTRO-H で計画されている衛星時刻 (TI) は要求時間分解能を達成できない。そこで前章で、より高い時間分解能の時刻をイベントにつける方法を提案した。我々は、その次のステップとして、実際にこの枠組みで時刻配信と時刻付けを行うためのシステムを ASTRO-H を模擬した地上試験用エレクトロニクスに構築した。この章では、その詳細について述べる。

5.1 実装方法の概要

時刻付け機能のうち、タイミングを厳密に規定すべき箇所は、CPU によるソフトウェア処理を介さないほうがよいので、FPGA(次節 5.1.1) 搭載の MIO board (section 3?) に搭載する。それ以外の SpacePacket 生成等の機能は CPU 基板 SpaceCard の機能として提供される。本節では MIO board に搭載する FPGA のハードウェアロジックについての開発を行う。したがって、時刻付けに関する地上試験を行うためには、FPGA に機能を実装する必要がある。この節では、FPGA を用いた実験のおおまかな流れを述べる。

5.1.1 Field Programmable Gate Array (FPGA)

FPGA は、人が自由にハードウェア・ロジックを設計できる「デジタル回路」をもった素子である。高性能でさまざまな用途の論理回路を、簡単に設計できるのが魅力である。

まず、デジタル回路といえば、複数の同一のゲートを備えた IC(Integrated Circuit) があげられる。数種類の IC を大量にならべればどんな論理の回路も作成することができる。この方法は単品が安価であるが、プリント基板に並べていちいちハンダ付けしなければならず、実装面積が大きくなる、など、手間や時間のかかる面で難がある。高信頼性が要求される衛星の信号処理部では、電源制御等の一部の回路には　IC と 抵抗やコンデンサーによる実装が施されるが、ミッションにクリティカルでない部分は、実装面積や設計の自由度等の利点から、FPGA 等の集積回路がよく利用される。

近年、主流となったデジタル設計は、デスクトップ上で自動化 (EDA : Electorical Design Automation) されている。すなわち、EDA ツールでシミュレーションと論理合成 (実際の設計の実装に置き換える作業) を行い、ハードウェアに実装するのである。これらハードウェアは、たいてい CMOS[*1] テクノロジを利用し、高密度でプログラム可能なロジックをもっている。単品は IC よりもはるかに高価であるが、迅速な論理の実装とデバッグ作業の魅力のほうが上回る場合が多い。

プログラム可能なハードウェアの中で最も複雑なものが、Field Programmable Gate Array (FPGA) である。FPGA の中の構造は、製造会社によって違う。そのため、専用の EDA ツールを使う必要があ

[*1] Complex MOS (Metal Oxide Semiconductor) に利用される技術の総称。半導体技術を応用することで、トランジスタやインバータを実現でき、AND, OR など各種ゲートを作り出すことができる。

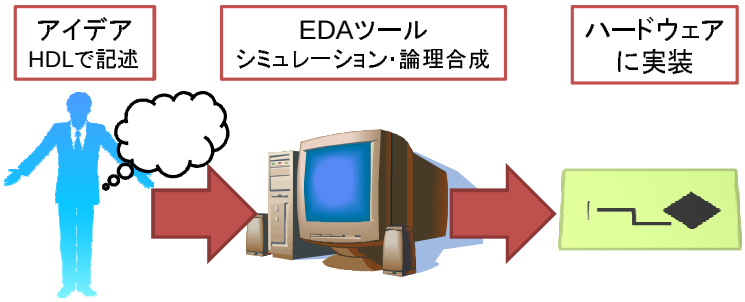

図 5.1　自動化されたデジタル設計のイメージ

る。例えば、本論文で扱う Xilinx (ザイリンクス) 製の Spartan-3 というタイプの FPGA は、論理セルのアレイで構成されていて、PROM というフラッシュメモリを使ってプログラムされる。EDA ツールは ISE という Windows 用ソフトウェアである。

FPGA は、宇宙用に使われ始めた当初、信頼性が低く、衛星軌道上の放射線環境下では寿命も短かった。だが、最近では、FPGA の信頼性も高くなっており、宇宙機用にも使えるものが増えてきている。現在では、ほとんどの人工衛星や科学衛星のハードウェアに FPGA が搭載され、実際に軌道上で成果を上げている。

IP core の一つには、 SpaceWire や RMAP の規格に基づいたロジックが SpaceWire IP core として提供され、ユーザーが自由に使える形になっている。ただし、MIO board 等の衛星搭載品には、担当業者が SpaceWire 等の規格にのっとり、衛星用として開発・品質保証したロジックが搭載される。また、MIO board の FPGA のみ、光子イベントへの波高値解析、画像処理、時刻付けといった機能も実装される。

こうした機能を FPGA に実装するためには、ハードウェア記述言語 (Hardware Description Lauguage : HDL) などを用いて回路の論理を記述しなければならない。次の節では、HDL の概要を述べる。

5.1.2　ハードウェア記述言語

FPGA に論理を実装するには、人が論理を EDA ツールに教える必要がある。そのための方法として、回路図を書いて EDA ツールに読み込ませる方法と、ハードウェア記述言語 (HDL) を使って論理をプログラムする方法がある。現在は、後者の方法が主流となっている。

はじめ FPGA 製造会社ことに 1 種類の HDL を使っていたため、他の FPGA に実装する際は人は言語の書き換えをしなくてはならなかった。現在、HDL はおもに Verilog HDL と VHDL の 2 種類に統一されている。EDA ツールは FPGA ごとに違えど、言語が同じなので、FPGA を変えるごとに言語の書き換えをする手間がなくなった。

本実験では、VHDL を用いてハードウェアの論理を記述する。

5.2　実験機器

3 章で紹介した ASTRO-H 用の SpaceWire 搭載エレキ、SpaceCard と MIO board は、まだ開発段階にある。本修士論文では、これらに代替する商用の SpaceWire 搭載エレクトロニクスとして下記を使用する。

- **SpaceWire Digital I/O board** : CPU なし、FPGA 搭載。MIO board および SMU の時刻配信部の代役として使用
- **SpaceCube 1** : ルーターおよびデータ読出し (SpaceCard の代用) として使用

を紹介する。

5.2.1 SpaceWire Digital I/O board

SpaceWire Digital I/O board(以下 DIO board) は、シマフジ電機が JAXA と共同開発した SpaceWire 搭載用の FPGA ボードである。主に衛星搭載前の (地上における) 開発段階で使われる。本実験では、SMU (SpaceCube2) の中の時刻配信機能を持つ部分 (Time Master) と、観測系のノンインテリジェント・ノード (MIO board / User Node) の代わりとして、DIO board を用いた。

DIO board の写真と搭載エレクトロニクスを図 5.2 に示す。ボードには、2 個の FPGA が搭載されている。これらはそれぞれ、SpaceWire FPGA、User FPGA と呼ばれる。SpaceWire FPGA は SpaceWire や RMAP のインターフェースに関するロジック (IP コア) を、User FPGA はユーザーが自由に実現したい論理を、それぞれ実装するために搭載されている。このように二つに FPGA を分けることで、どのシステムでも共通な機能と、システムごとに変えなければならない機能を明確に分離している。

購入時、SpaceWire FPGA には、シマフジ製の SpaceWire IP コアと RMAP IP コア (RMAP 仕様に基づいた通信のためのロジック)、SDRAM[*2]コントローラ (SDRAM への読み書きを制御する) が実装されている。User FPGA には、CMOS I/F、LVDS I/F、LED、Dip スイッチが User FPGA に接続されていて、ユーザーは User FPGA からこれらの信号を入出力として使うこともできる。(例えば、特定の信号をオシロスコープで見たり、ある状態に遷移したら LED を点灯させることができる) また、SpaceWire FPGA を経由すれば、User FPGA から SDRAM にデータを記録したり、SpaceWire リンクへパケットを送ることもできる。

5.2.2 SpaceCube 1

SpaceCube 1 は、シマフジ電機が JAXA と共同で、SpaceWire の地上実験向けに開発した小型コンピュータである。本実験では、ルーターとして、ルーター機能付き IP コアを実装した SpaceCube 1 を使用した。また、次節で紹介する SpaceWire Digital I/O board からデータを読み出す際にも同じ SpaceCube 1 を用いた。

SpaceCube 1 は図 5.4 のような外観をしている。SpaceCube 1 の性能を表 5.2 にまとめた。内部構造は、図 5.5 ようになっている。SpaceWire 通信をするための SpaceWire のポートは三つあり、そのコネクタには、3 つの Dsub 9pin を採用している。FPGA には、シマフジ製の SpaceWire IP コア (SpaceWire 仕様に基づいた通信のためのロジック) が実装されている。搭載 OS(Operating System) は、TRON プロジェクトの一環で作られたリアルタイム OS の「T-Kernel」である。SpaceCube 1 は大変小さいながらも、ディスプレイやキーボードなどを接続すればふつうの PC と同じように使うことができる。

SpaceCube 1 の FPGA 部分をルーター機能つきの SpaceWire IP コア (ルーター IP コア) にし、CPU 上でルーター用ドライバを動かすと、外部 SpaceWire インターフェース 3 つが CPU 経由でスイッチングされてルーターのようにつながる。

[*2] 揮発性の記憶装置。多くのパソコンのメモリに使用される

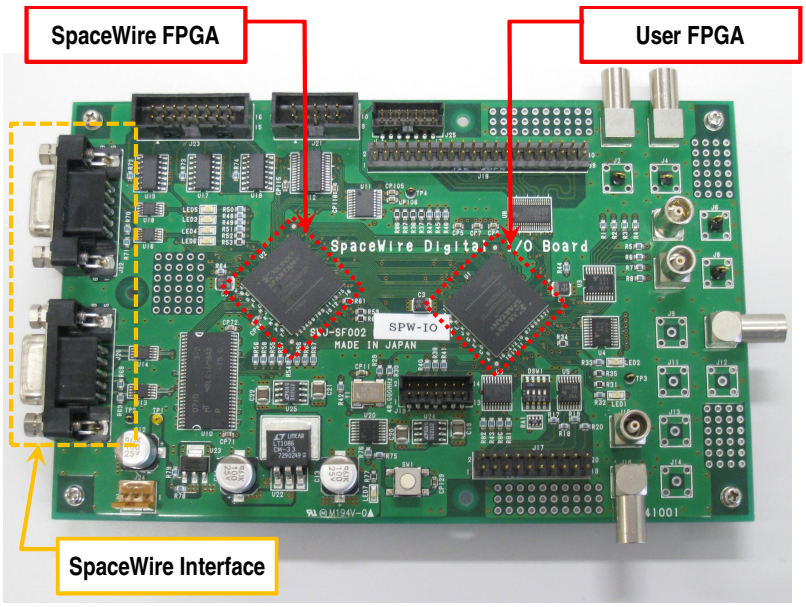

図 5.2　SpaceWire Digital I/O board の写真と各搭載エレクトロニクス

インターフェース	SpaceWire (Dsub 9pin), LVCMOS IN 8bit / OUT 8bit, LVDS IN 12bit / OUT 12bit, RS232C, RS422 JTAG × 2
その他	FPGA : Xilinx Spartan-3 XC3S1000 × 2 SDRAM : 16 MBytes OSC clock : 48 MHz
電源	DC +5V

表 5.1　SpaceWire Digital I/O board の性能

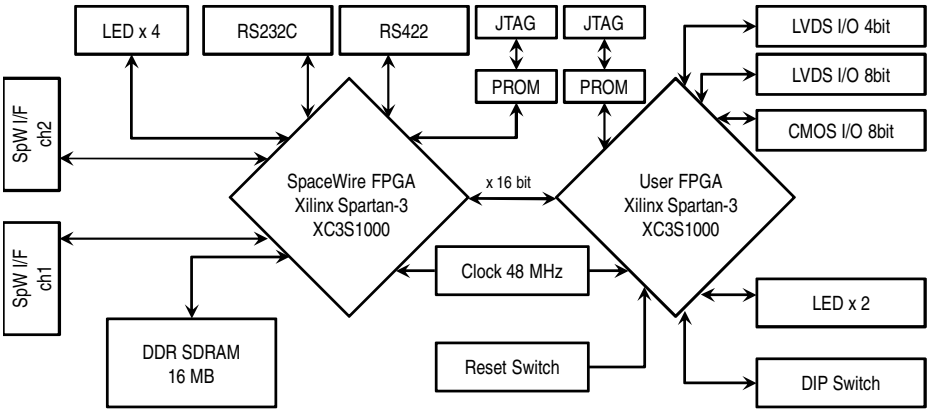

図 5.3　SpaceWire Digital I/O board のブロックダイアグラム (シマフジ電機による SpaceWire Digital I/O board 仕様より作成)

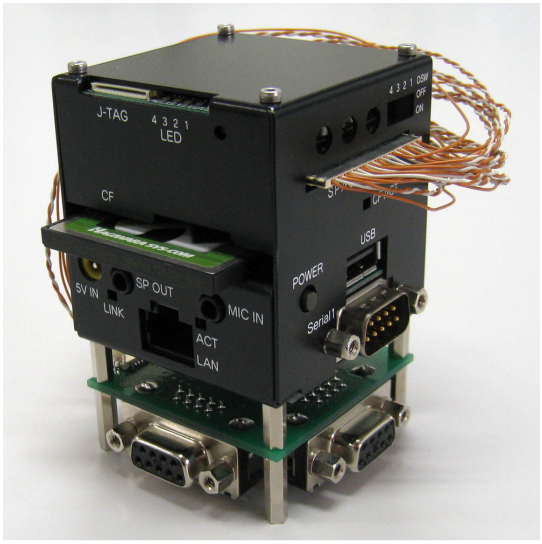

図 5.4　SpaceCube 1 の写真

CPU	VR5701, 200 MHz
RAM	64 MBytes
Flash Memory	16 MBytes
インターフェース	SpaceWire (Dsub 9pin) × 3, Compact Flash (True IDE), XGA video out (1024 × 768), Ethernet (100Base), Audio(Stereo), RS232C, USB 1.1, JTAG (Debug/FPGA)
電源	DC +5V
サイズ	52 × 52 × 55 mm^3

表 5.2　SpaceCube 1 の性能

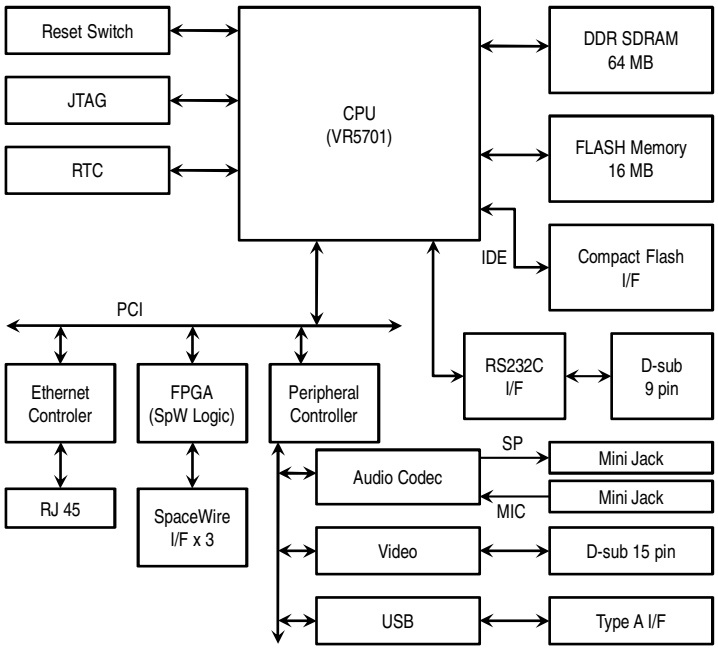

図 5.5　SpaceCube 1 のブロックダイアグラム。(シマフジ電機による SpaceCube 1 の仕様および [24] より作成)

5.3　実装した機能

私は、ハードウェア記述言語 VHDL を用いて時刻配信機能 (Time Master 機能) と User Node における時刻照合およびイベント時刻付け機能を作成し、2 枚の DIO board の FPGA に実装した。この節では、その機能を詳しく述べる。

2 枚の DIO board にはそれぞれ、Time Master として時刻配信を司る機能を持つ SMU、User Node として時刻付けを司る機能を実装させる MIO board「User Node」の役割をおわせる。前にも述べたように、まず Time Master (18 ページ) で 64 Hz の Time-Code を生成し、SpaceWire リンクを経由して User Node へ送信する。User Node はフリーランクロックの時刻カウンタ LocalTime (4.1.2 節, 21 ページ) をもつ一方、受け取った Time-Code をもとに TI を合成し、時刻の HK (TI と LocalTime の照合表) を SDRAM に記録していく。また、User Node は、イベントが入力されると、そのときの LocalTime を SDRAM に記録する。

すべての機能をまとめてひとつの VHDL ファイルに記述するのは大変なので、通常、個々の機能は個々の「モジュール」という単位に分けて記述する。図 5.6 に、実際に、本修士論文で設計したモジュール間の機能ブロック図を示す。

5.3.1　Time Master 機能

Time Master は、SpaceWire Time-Code を生成して配信する。SMU の場合はそれだけでなく、TIME DATA も生成して RMAP write で配信するのだが、本実験では簡単のため TIME DATA の生成・配信を省略した。

Time-Code の生成には GPS クロックを使った。我々が用いた GPSR は、古野電機製 GF8050 で、出力されるクロックは 1 pps と、10 MHz であった。Time Master となる DIO board の CMOS I/F を通してこのクロックを User FPGA に入力した。(詳細は第 6 章で述べる)

Time Master の詳しい FPGA モジュールの構造を図 5.6(上段) に示した。まず、CMOSIn から GPSR からのクロック (10 MHz) を入力する。TimeCode64HzSendModule がこのクロックを分周して 64 Hz で 1 カウントアップする Time-Code を生成し、SpaceWire FPGA に送る。SpaceWire FPGA には、単に SpaceWire IP コアのみをが実装されており、Time-Code の tick 信号とデータが SpaceWire の port から配信される。[*3]。

5.3.2　User Node における時刻付け機能

User Node(18 ページ) は、Time Master から受け取った Time-Code と TIME DATA を合成して TI を合成する。しかし、本実験では、Time Master での TIME DATA の生成・配信を省略したため、User Node で Time-Code を元に擬似的な TIME DATA を作り TI を合成した。また、高時間分解能を達成するため、User Node にはフリーランクロックから高時間分解能の時刻カウンタ (LocalTime) を生成する機能も持っている。

具体的な User Node の動作は、次のような流れになる。

[*3] 購入時のシマフジ製 IP コアから不必要な機能を削除している。

まず、配信された Time-Code から TI を生成する流れは、以下の通りである。
(1) TICounterModule は Time-Code が 0 (ゼロ) のタイミングで 1 カウントアップする時刻カウンタ「TIME DATA」を生成する。
(2) カウントアップしたら、すぐに TIRegisterModule の「次の TIME DATA 用レジスタ」に TIME DATA を書きこむ。書き込みは内部バス経由で行う。
(3) TIRegisterModule は Time-Code が 3 のタイミング[*4] で「今の TIME DATA 用レジスタ」に「次の TIME DATA 用レジスタ」の値をコピーする。

一方で、高分解能の時刻をつけるために、下記の二つの機能を実装した。
(4) (1)〜(3) の間も、LocalTime を生成している。
(5) Time-Code が 0 (ゼロ) のとき TI (TIME DATA + Time-Code) と LocalTime をラッチ[*5]し、外部バスを経由して SDRAM に書き込む。

また、イベントへの時刻付けとして、下記の機能を実装した。
(6) CMOSIn から信号を受けると、その時の LocalTime をラッチして、外部バス経由で SDRAM に記録する。

User Node の SpaceWire FPGA と User FPGA の両者に外部バス (External Bus) が備わっており、両者の間のデータはバス経由でやりとりされる。User FPGA の各モジュール間のデータは内部バス経由でやりとりされる。User FPGA のロジックは、日本 SpaceWire ユーザー会による UserFPGATemplate をもとに作成した。SpaceWire FPGA は、シマフジ製 IP コア[*6]と SDRAM コントローラ (SDRAMC) のコードに加えて、外部入出力 (aux) から Time-Code (Tick と 6 bit の時刻情報部) を User FPGA に出力する機能を追加した。これによって Time-Code を User FPGA で参照できるようにした。

[*4] このタイミングを選んだ理由は、TIME DATA レジスタと時刻照合表の書込みタイミングをずらすためである。Time-Code が 3 で「次」から「今」に書き換えるのは、ASTRO-H と同じ仕様でもある。
[*5] ある瞬間のレジスタなどの値をコピーしておくこと。バス経由でメモリ領域に書込みをするとき、その瞬間の値を記録したい場合は、カウンタなどの値が書込み時間中に変化してしまう可能性があるので、ラッチした値を書き込むようにする。
[*6] http://www.shimafuji.co.jp/を参照

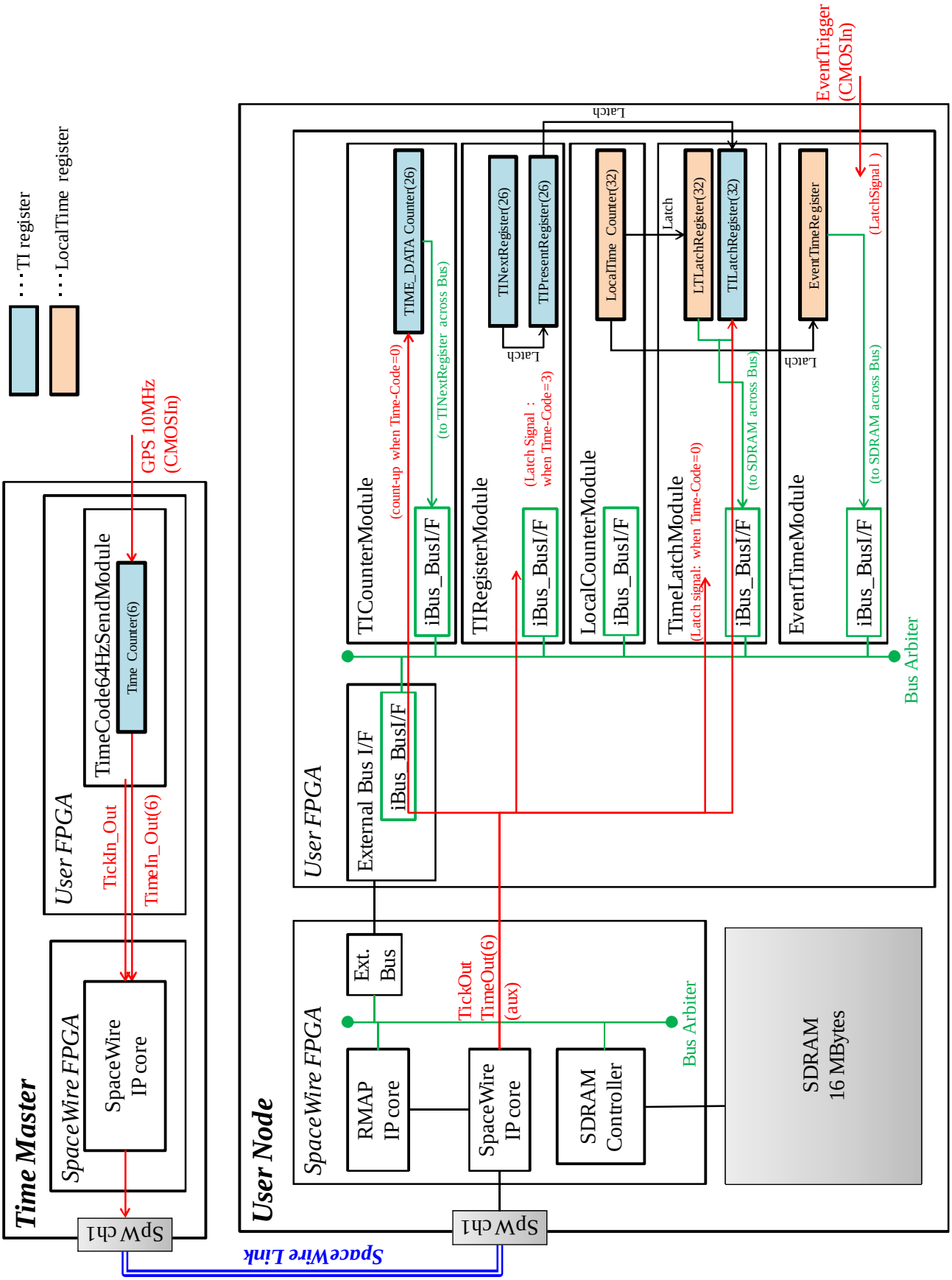

図 5.6 SpaceWire Digital I/O board の SpaceWire FPGA と User FPGA に実装したモジュールのブロック・ダイアグラム。矢印の緑は FPGA の内部のバスに関する信号および信号線、赤は FPGA の外部との入出力やインターフェースからの入出力を示す。水色の四角は TI に関するレジスタ、オレンジの四角は LocalTime に関するレジスタを示す。

5.4 時刻付けに用いる各機能の動作確認実験

前節で述べたように、我々は Time Master や User Node の FPGA ロジックを新たに設計した。そこで、この FPGA ロジックが期待される動作をしているか確認した。本節の実験の目的は、(1) Time-Master で期待される Time-Code と Tick が出力されているか、(2) User Node で Time-Code が受信できているか、さらに、(3) SDRAM に書き込むデータは正しく書きこまれているか、確かめることである。

5.4.1 実験セットアップ

Time Master と User Node 両者の SpaceWire port (ch1) を SpaceWire ケーブルで接続した。今回はロジックのみの確認実験であるため、GPS 受信機 (GPSR) は使用せず、代わりに GPSR からのクロックを模擬した 10 MHz の階段波を用いた。DIO ボードの CMOS Out から Tick 信号と Time-Code の時刻部分 6 bit を出力し、デジタル入力をもつオシロスコープで可視化した。

Clock Generator (TECHNOLAND CORP. N-TM 203) から出力される TTL 信号は振幅 +5 V であるため、DIO board の LVCMOS (LVTTL : +3.3 V) にそのまま入力できない。そのため、抵抗を間にはさみ電圧降下させ、3.3 V に調節した。

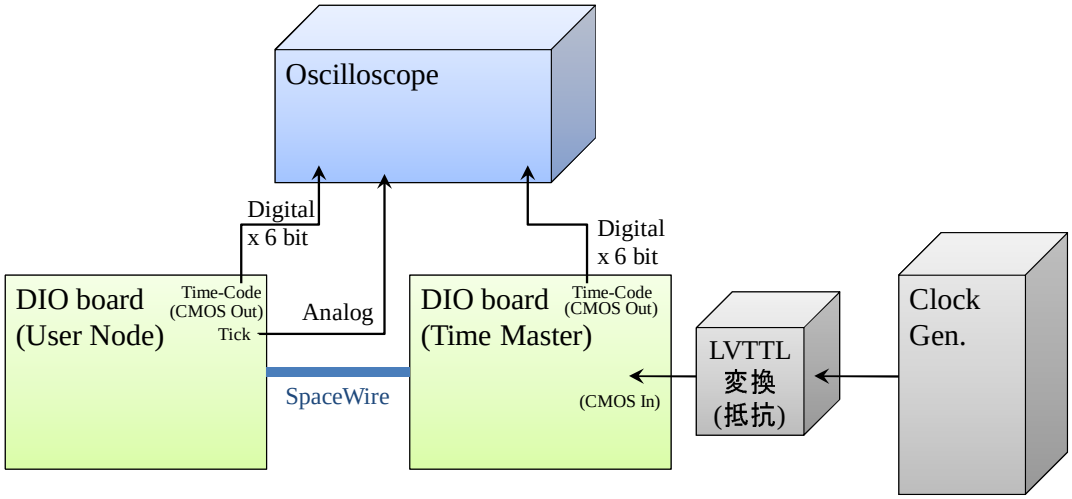

図 5.7 確認用の実験セットアップ。GPS 10 MHz クロックの代わりに 10 MHz の矩形波発生装置を使用した。

5.4.2　結果

前述の目的 (1),(2),(3) を行うためには、それぞれ順に下記の (a),(b),(c) の実験を行う必要がある。この節では、それぞれの実験結果をまとめる。この節では、(a) Time Master：Time Code を 64 Hz で送信、(b) User Node：Time Code を 64 Hz で受信、(c) User Node：TI と LocalTime の比較表の記録を確認した実験の結果をまとめる。

(a) Tick と Time-Code の出力確認

まず、Time Master の「TimeCode64HzSendModule」が、期待される Tick と Time-Code の時刻を生成していることを確認した。図 5.8 はオシロスコープで確認した Tick と Time-Code の中身のようすである。Tick は Time-Code が送信されるときに “1” であることが規定されていた (2.2 節) が、そのとおりに出力されていた。

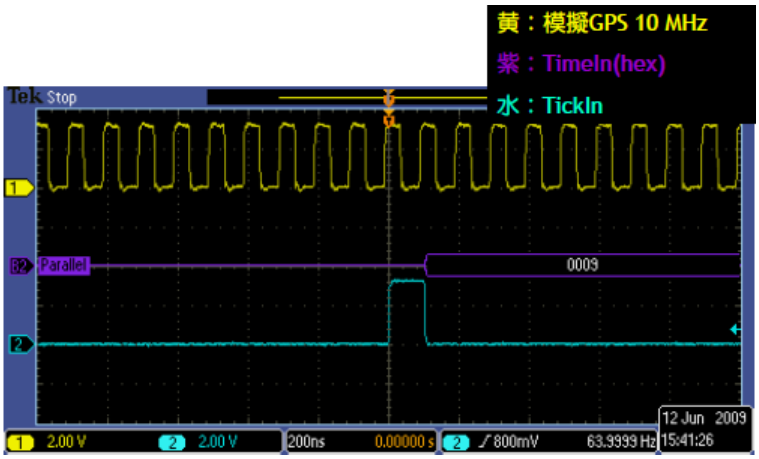

図 5.8　Time Master の Tick と Time-Code の出力タイミングの確認。黄色は模擬 GPS 10 MHz クロックを、水色と紫色がそれぞれ User FPGA が出力している Tick と Time-Code の時刻 (16 進数) を示している。

次に、Time-Code の時刻がおよそ 15.6 ミリ秒周期すなわち 64 Hz で 1 ずつカウントアップし、16 進数で 0x00 – 0x3F(10 進数で 0〜63) まで回りきったら、次はまた 0 に戻ることを確認した。図 5.9 はオシロスコープで確認した Time-Code のカウントアップのようすである。Time-Code が 16 進数で 0〜3F(10 進数で 0〜63) までカウントアップし、次はまた 0 に戻っていた。また、64 Hz で安定した Time-Code が出力されていた。

(b) Time Master で生成した Time-Code が User Node で正しく受信されていることの確認

さらに、Time Master から配信された Tick と Time-Code の時刻が、UserNode で、正しいタイミングで正しい時刻データが受信できているかを確認した。図 5.10 はオシロスコープで確認した Time Master と User Node の Time-Code カウントアップのようすである。両者ともに Time-Code が 16 進数で 0x00 – 0x3F(10 進数で 0〜63) までカウントアップし、次はまた 0 に戻っていた。また、およそ 15.6 ミリ秒周期すなわち 64 Hz で Time-Code が送受信できており、そのタイミングも同じであった。

Time Master のほうが User Node よりも Time-Code が 1 大きいのは、Time Master の User FPGA から出力されている TimeIn は次の Tick で出力する Time-Code と、User Node の User FPGA から出力されている TimeOut は受け取った Time-Code とを比較しているからである。つまり、Time Master

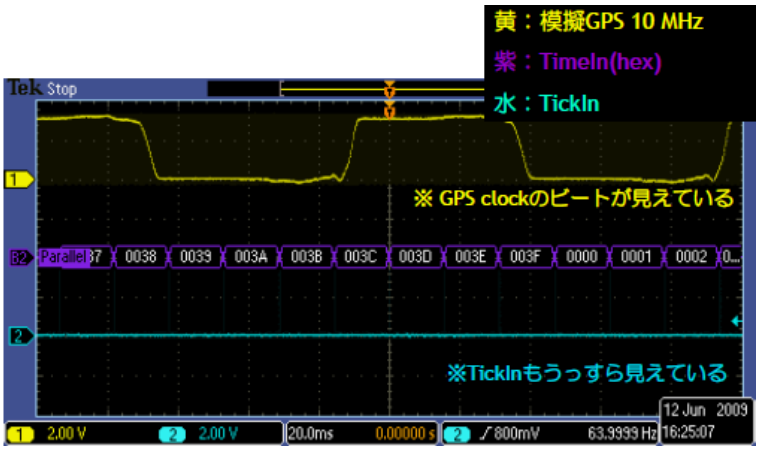

図 5.9　Time Master における Time-Code のカウントアップ確認。色の別は図 5.8 と同じ。黄色はクロックより大きい周波数のビートが見えている

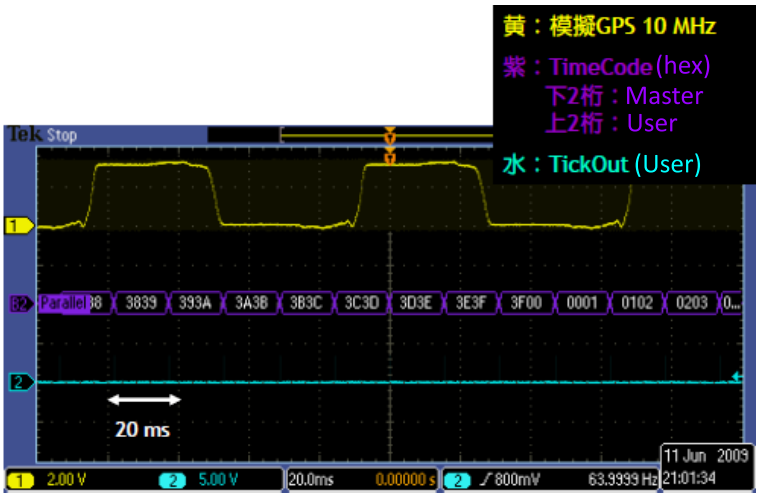

図 5.10　Time Master と User Node の Time-Code を比較したもの。黄色は模擬 GPS 10 MHz クロックを、水色が User Node の User FPGA が SpaceWire FPGA から受信した Tick を示している。また、紫色の上 (左) 二桁が Time Master の User FPGA が出力した Time-Code を、下 (右) 二桁が User Node の User FPGA が受信した Time-Code の時刻 (16 進数) を示している。黄色はクロックより大きい周波数のビートが見えている

と User Node で送受信されている Time-Code の値は同じだったことが確認できた。

(c) User Node で TI と LocalTime の照合表が SDRAM に記録されていることの確認

最後に、User Node で、TI と LocalTime が正しく生成され、これらを照合したデータが正しく SDRAM に書き込まれていることを確認した。

図 5.11 は、SDRAM を SpaceCube 1 を用いて読みだしたデータを Windows PC のエミュレーター越しに見た画面である。TI は、Time-Code が 0(ゼロ) のときにラッチしたものを書きこんでいるので、2 進数にして下 6 桁がすべて 0(ゼロ) になっていた。また、上 26 桁は TIME DATA を示しており、1 ずつカウントアップしていた。

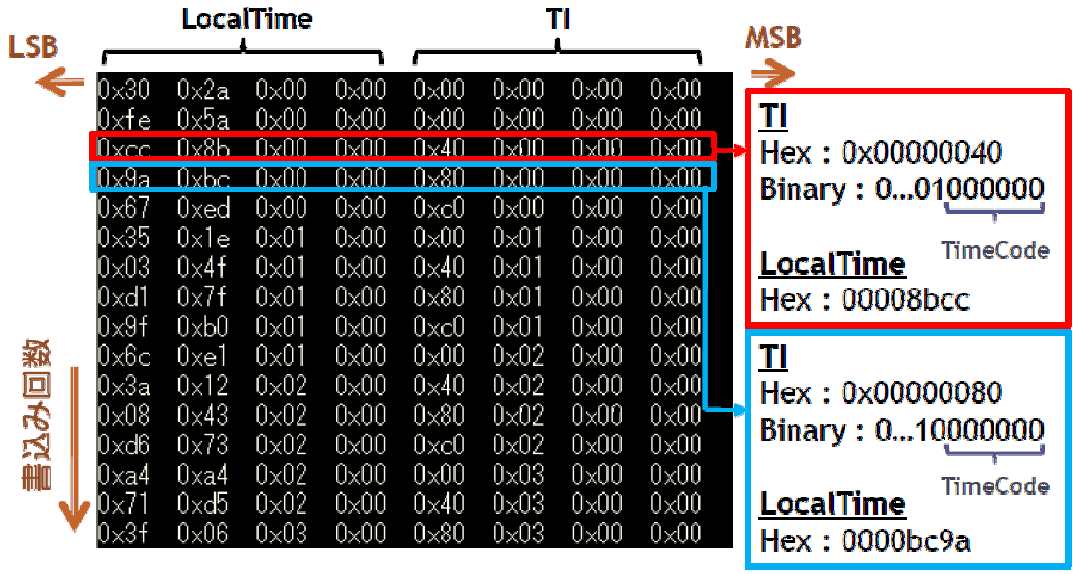

図 5.11　TI と LocalTime の照合データを SDRAM から読みだしたところ。

第 6 章

時刻精度の測定

これまで、ASTRO-H 衛星における時刻付け手法を SpaceWire 搭載エレクトロニクスに実装する方法と、その機能の動作確認を行った結果を示した。この章では、この時刻付け手法における時刻精度を悪化させる、(1) Time-Code のジッタ、(2) ローカルクロックの時刻安定度を実測した実験について詳細に述べる。さらに、(1) と (2) をトータルで考えた時刻精度を測定した実験について述べる。最後に、実験結果から、ASTRO-H に要求されている時刻精度を達成できるか議論する。

6.1 Time-Code のジッタ測定

6.1.1 概要

Time-Code のジッタは、4.2.1 節で述べたように、ASTRO-H 衛星の時刻付けにおいて User Node で再合成される TI の時刻精度を悪化させる。したがって、本論文で検証する時刻付け手法を用いたとき、イベントの時刻精度に影響する。ここでは、SpaceWire エレクトロニクスに ASTRO-H と同様の時刻配信機能を実装し (前章参照)、Time-Code のジッタを測定する実験について述べる。

6.1.2 実験セットアップと実験方法

図 6.2 は、実験のセットアップである。10 MHz は TTL 規格 (0 〜 +5 V) の正弦波だが、DIO board に入力するには LVTTL 規格 (0 〜 +3.3 V) の矩形波に成形する必要がある。そこで、Discriminater (LeCroy MODEL 623B) で NIM 規格の階段波に成形し、さらに、Gate Generator (TECHNOLAND CORP. N-TM 203) で TTL 規格の矩形波に成形する方法をとった。Time Master は成形された 10 MHz クロックから 64 Hz でカウントアップする Time-Code を生成し、SpaceWire ポートへ送信する。このとき生成された Tick は CMOS Out から出力される。User Node は、SpaceWire リンクを通して Time Master が生成した Time-Code を受け取り、内部信号の Tick(TICK_OUT) を CMOS Out へ出す。Time Master と User Node から出力された Tick は、Gate Generator (Phillips Scientific MODEL 794) を通して LVTTL 規格から NIM 規格に変換されて、Time-to-Analog Converter(TAC)、そして Pocket MCA へ入力される。TAC や Pocket MCA は、Time-Code のジッタを測るのに重要であるので、42 ページに詳細を述べる。

第 5 章でも述べたように、SpaceWire 搭載エレクトロニクスとして SpaceWire Digital I/O board (DIO board) と SpaceCube 1 を用いた。Time Master は、第 5 章で述べたままであるが、User Node の FPGA では、極力 Time-Code 以外のジッタが入らないよう、Time Master と同じ内容の SpaceWire FPGA から受信した Tick をそのまま CMOS Out に出力するだけの構造になっている。また、本実験では SpaceWire のリンクレートは 100 MHz に設定されていた。

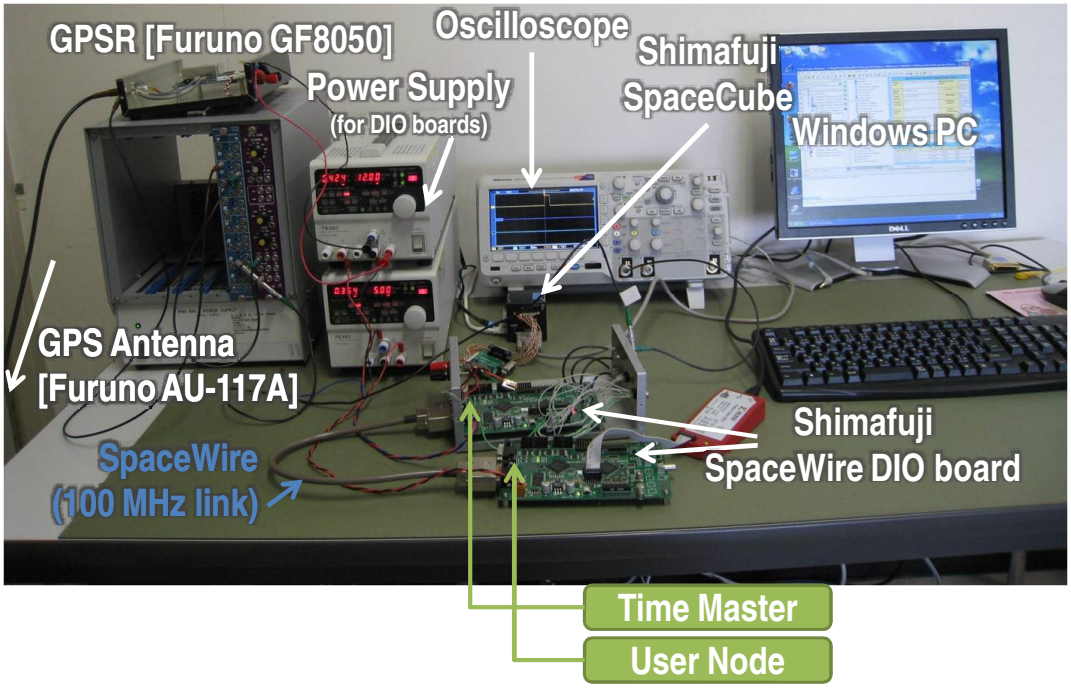

図 6.1　実験セットアップの写真

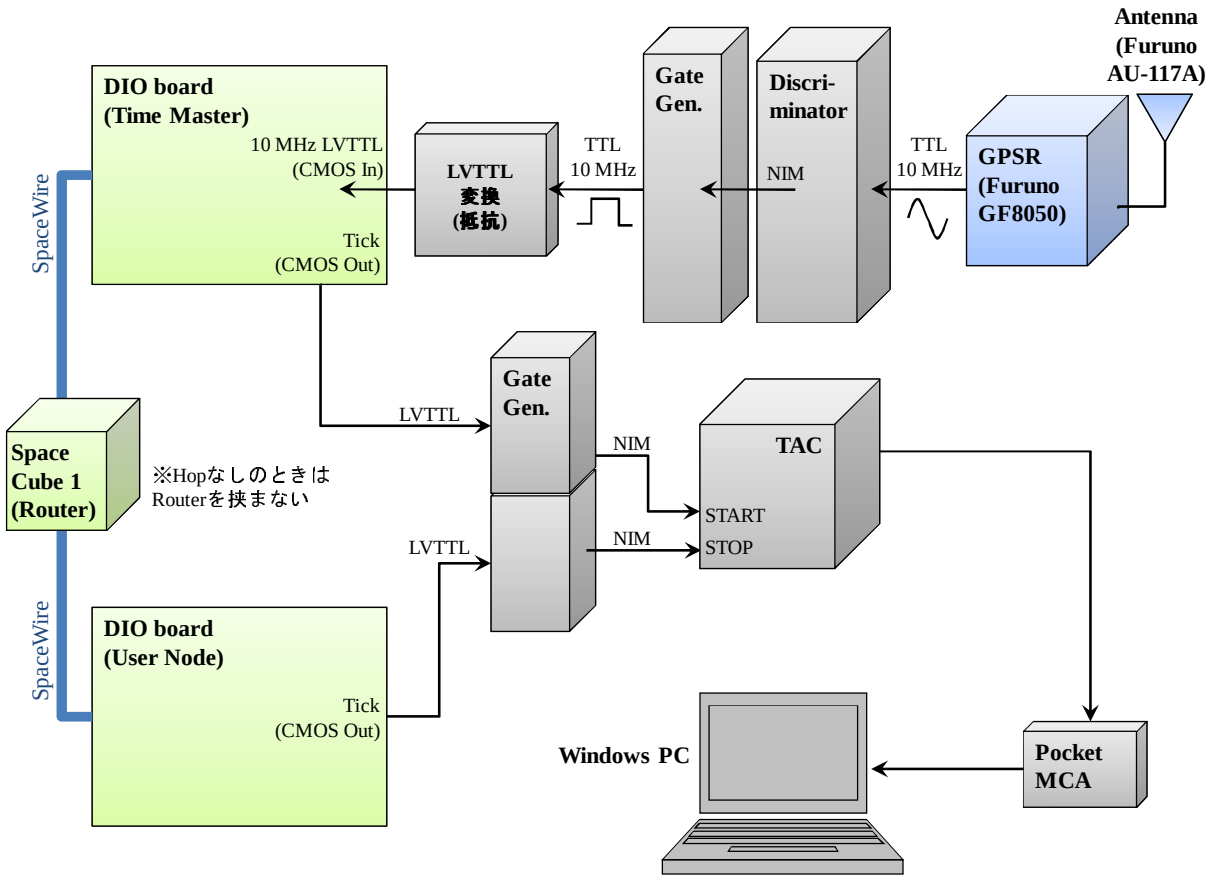

図 6.2　Time-Code のジッタ測定セットアップ

GPSR および GPS アンテナ

過去のX線天文衛星の地上実験で実績のある古野電機製の GPSR を使用した。我々が実験に用いたのは、GPSR : Furuno GF8050 (図 6.3 左) と GPS アンテナ : Furuno AU-117A (図 6.3 右) である。

Furuno GF8050 は、恒温槽付発振器と、クロック生成機能をもつ FPGA から構成される。後者の FPGA には、GPS アンテナより受信した正秒のタイミング (1 pps) から 10 MHz クロックを生成するロジックが焼きこまれている。電源投入後、GPS アンテナから GPS 衛星の電波を受信し、一定時間 (約 30 分〜1 時間)GPS 衛星を捕捉した状態が続くと、出力クロックの時刻精度 (< 300 nsec) が安定してくる。もし GPS 衛星を 1 機も捕捉できない状態が続いても、FPGA ロジックが GPS 時刻のトレンドからおおよその補正をする仕組みになっている。どの状態に遷移したかどうかは、基板に配置された LED の点灯パターンからわかるようになっている。

アンテナは、ベランダに出し、できるだけ広い空をとらえられるように設置した。図 6.4 に示すように、現在稼働中の GPS 衛星は東京の全天で同時に 10〜15 台程度見える。しかし、天候が荒れると GPS 衛星の電波が厚い雲にさえぎられてしまう。このような場合、GPSR が高い精度のクロックを出せない。我々は、つねに GPSR から高い精度のクロックを得るため、晴天もしくは曇天の日のみ実験を行った。

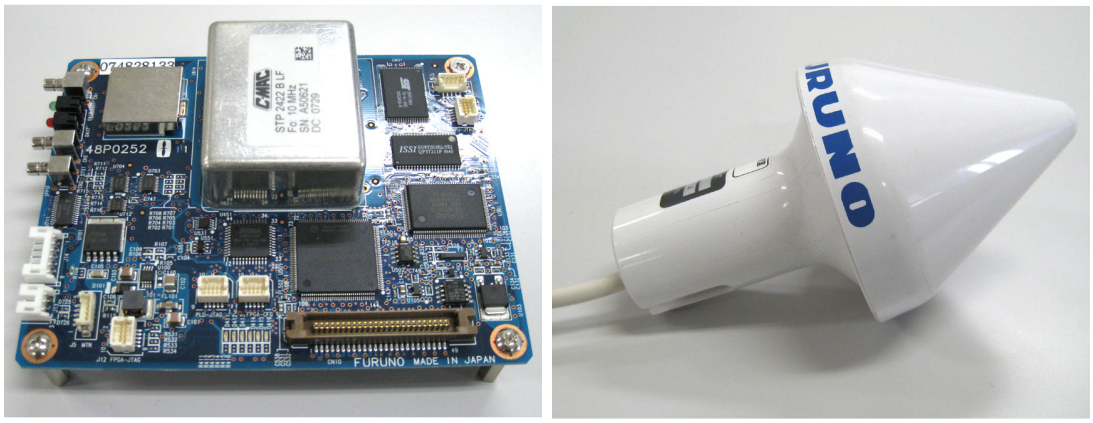

図 6.3 (左) GPSR : Furuno GF8050、(右) GPS アンテナ : Furuno AU-117A

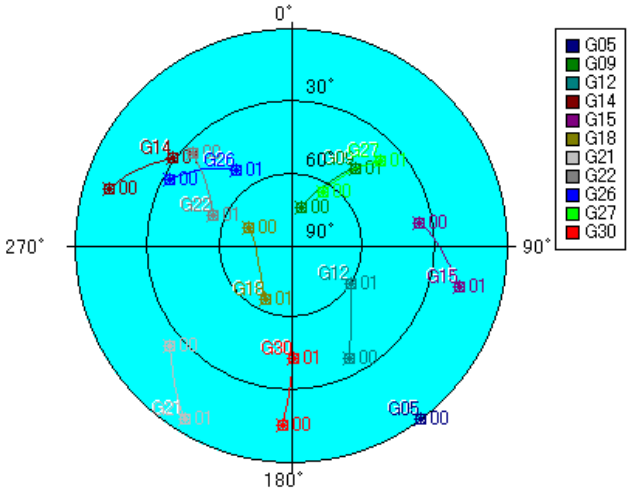

図 6.4 ある日の GPS 衛星の軌道解析。東京の上空 (水色円内) で各 GPS 衛星が 1 時間のあいだにどのような軌道をたどるかを示している。凡例は GPS 衛星の番号を示す

TAC (Time-to-Analog Converter)

TAC は、Time-to-Pulse-height converter とも呼ばれ、2 つの信号の時間差を電圧に変換する。

我々が用いたのは、ORTEC 467 Time-to-Pulse Height Converter/SCA で、“START” と “STOP” に入力した信号の時間差を電圧に変換する仕組みになっている。このとき、電圧と時間差は線形の相関をもっている。電圧と時間差の関係を変えるパラメータとして、Range (測定可能な最大時間幅)、Multiplier (倍数) がある。たとえば我々の実験では Range 0.4 μsec、Multiplier $\times$10 だったので、信号の時間差 Δt と TAC の出力信号 V_{TAC} の関係は、

$$\frac{\Delta t}{V_{TAC}} = \frac{0.4 \times 10\ \mu\text{sec}}{\text{Max 10 V}} \tag{6.1}$$

となる。ただし、最大値付近は電圧と時間差の線形性が低い。実験ではこのことを考慮して、適切な Gain と Multiplier を選んだ。

Pocket MCA

MCA (Multi Channel Analyzer) は、ある一定の電圧幅をひとつのチャンネルとして、電圧値をチャンネルという単位に変換する。チャンネルと電圧は線形の相関をもつ。我々は、Amptek Pocket MCA 8000A を用い、Amptek ADMCA というソフトウェアでチャンネルを横軸とした TAC 出力の電圧値のヒストグラムを作成した。我々の実験での ADMCA および Pocket MCA の設定から、MCA に入力される信号電圧 V_{MCA} と電圧と MCA チャンネル (ch) の比は、

$$\frac{\text{Channel}}{V_{MCA}} = \frac{\text{Max 4096 ch}}{\text{Max 5 V}} \tag{6.2}$$

であった。(6.1) 式と (6.2) 式から、TAC に入力した 2 信号の時間差を MCA のチャンネルの値 $N_{Channel}$ から計算すると、$V_{TAC} = V_{MCA}$ なので、

$$\Delta t = \frac{\Delta t}{V_{TAC}} \div \frac{\text{Channel}}{V_{TAC}} \times N_{Channel} \simeq 0.49\ \text{nsec} \times N_{Channel} \tag{6.3}$$

6.1.3 結果

ルーターなし (以下、hop なし) と、1 つルーターを挟んだとき (以下、1 hop あり) それぞれの Time Master と User Node の Tick の時間差を、前に述べたようなセットアップで測定した。なお、測定中、SpaceWire リンクでは、データ送信を行わなかった。すなわち、Time-Code 以外で通信されるパケットは NULL のみであった。

図 6.5 は、MCA から得られた時間差のヒストグラムである。6.3 式を用いて MCA チャンネルから時間差に変換した。ジッタは図 6.5 では、ヒストグラムの裾の左端から右端までの x 軸方向の距離として計算すると、hop なしと 1 hop ありのときの固定遅延とジッタの値は、表 6.1 のようになった。

	固定遅延 (nsec)	**ジッタ** (nsec)
hop なし	160	90
1 hop	410	160

表 6.1　遅延時間とジッタの測定値

6.1.4 考察

Time-Code のジッタ値

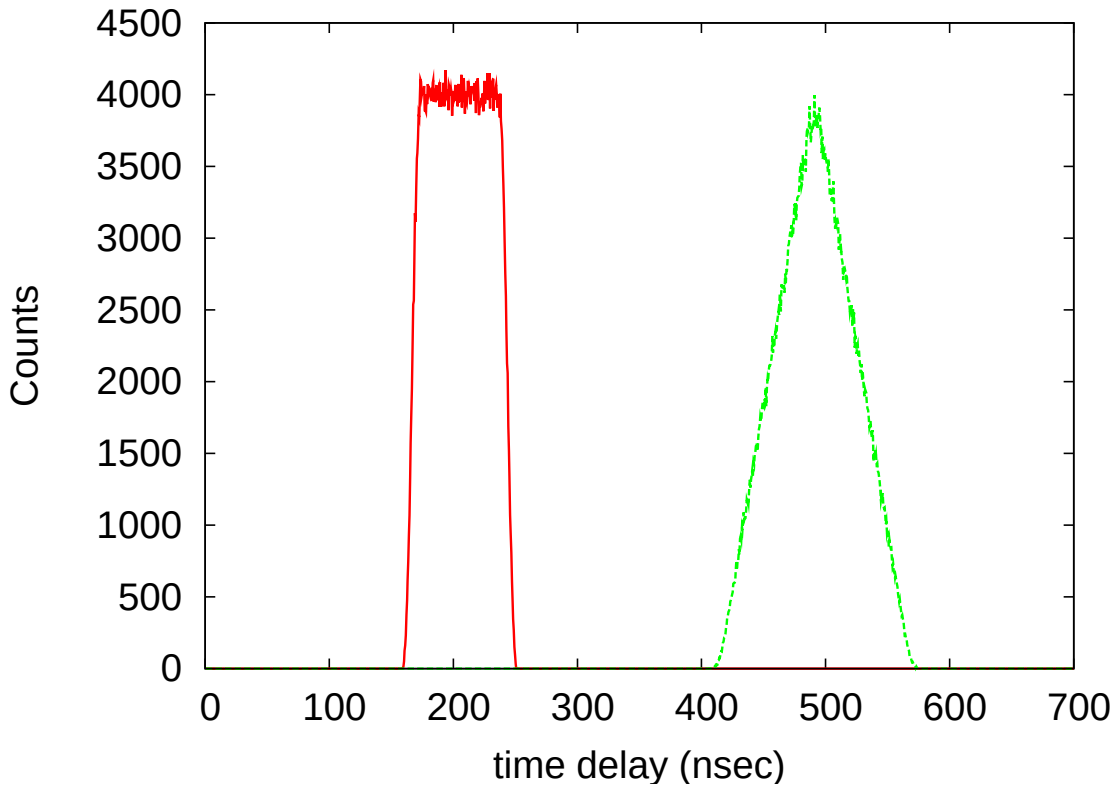

図 6.5 ジッタ測定の結果を示すヒストグラム。赤は hop なし、緑は 1 hop あり。

Time-Code のジッタは、既に第 4 章で、no hop のときに 80 nsec、1 hop のときは 2 倍の 160 nsec と見積もった。実験結果は、no hop のときだけ 10 nsec 大きく、no hop のときにジッタは 1 hop の 2 倍になっていなかった。原因として、Time-Code のジッタのほかに、FPGA 内で生じる信号線由来の時刻揺らぎ (これもジッタと呼ぶことにする) が TAC で測定されていることが考えられる。本来は、Time-Code のジッタそのものに近い値を測るためには、Time Master と User Node 両者の SpaceWire FPGA から出力される Tick を TAC に入力しなければならなかった。しかし、今回は技術的な問題から、Time Master では User FPGA で生成した Tick を使用した。そのため、純粋な Time-Code のジッタのみを測ることができなかった可能性がある。

ここで、hop なしのときの Time-Code のジッタを σ_{tc}、その他のジッタを σ_{other} と置く。Time-Code のジッタは n hop のとき $(n+1)\sigma_{tc}$、σ_{other} は hop の数に限らず同じだと仮定すると、本実験セットアップの TAC で観測される全ジッタ σ_{all} は、

$$\text{hop なし}：\sigma_{allnohop} = \sigma_{tc} + \sigma_{other}$$

$$1\text{ hop}：\sigma_{all1hop} = 2\sigma_{tc} + \sigma_{other}$$

上の 2 式の σ_{all} に表 6.1 の実測値を代入して解くと、

$$\sigma_{tc} = 70\text{ nsec},\ \sigma_{other} = 20\text{ nsec} \tag{6.4}$$

である。よって、hop なしのときの Time-Code のジッタは 70 nsec、1 hop のときの Time-Code のジッタは 140 nsec である。

Time-Code は NULL (第 2 章参照) の通信中に送信されていた (24 ページ 図 4.7 の Data Character (10 bit) が NULL (8 bit) になったと考えればよい)。なので、1 bit 分のジッタは $70/8 = 8.8$ nsec となる。これが Time-Code に割り込みする Character のなかで、Time-Code 以外で最も大きい Data Character は 10 bit なので、予想されるジッタは 88 nsec となる。

ジッタのヒストグラムの確率分布

我々の実験では、Time-Code は NULL(8 bit) の通信中に 100 MHz リンクレートで送信されていた。この条件での Time-Code ジッタのヒストグラムの確率分布をシミュレーションすると、図 4.8 (25 ページ) のようになる。この確率分布の形状は、実測したヒストグラム (図 6.5) と良く合っている。

ASTRO-H における Time-Code ジッタ

我々の実験で、SpaceWire のリンクレートは 100 MHz であった。ASTRO-H の場合、電力などの観点から、SpaceWire のリンクレートは現在 20〜50 MHz になることが検討されている。Time-Code のジッタはリンクレートに反比例する (式 (4.1) を見よ) ので、リンクレートが遅い (値が小さい) とジッタは大きくなる。

ASTRO-H のリンクレートがネットワーク全体にわたって 20 MHz だったとき、1 hop ごとの Time-Code のジッタ (NULL のみ転送しているとき) は、100 MHz リンクにおける Time-Code のジッタが 70 nsec だったので、

$$70 \text{ nsec} \times \frac{1}{20 \text{ MHz}/100 \text{ MHz}} = 350 \text{ nsec}$$

さらに、Time Master (SMU) から User Node まで介するルーターの数も考えなくてはならない。SpaceCard で合成した TI を MIO board で参照して時刻 HK に用いるとすると、MIO board における TI の精度は SpaceCard で受け取る Time-Code のジッタに依存する。現在の計画では、SMU から SpaceCard までもっとも多くて 10 台のルーターを通るので、

$$350 \text{ nsec} \times 10 = 3.5 \ \mu\text{sec}$$

もし、SpaceWire リンクでかならずデータが通信されているとき (Data Character 10 bit の間を Time-Code が送信されるとき) には、

$$88 \text{ nsec} \times \frac{1}{20 \text{ MHz}/100 \text{ MHz}} \times 10 = 4.4 \ \mu\text{sec}$$

が SpaceCard で受け取る Time-Code のジッタになる。

6.2 時刻安定度の測定

6.2.1 概要

第 4 章で提案した非同期クロックを用いた時刻 tick 内挿法では、Time-Code のジッタ以外にも LocalTime の時刻安定度も時刻精度を悪化させる要因となる。LocalTime は MIO board の水晶発信器の周波数安定度そのものである。水晶発信器の周波数は、温度変化によって安定度が悪化する傾向にある。この現象は「温度ドリフト」と呼ばれている。そこで我々は、温度ドリフトに対して LocalTime の時刻安定度がどのように変化するか測定した。ただし、後述の測定手法上、Time-Code のジッタ込みでの測定となる。

実験はおおまかに以下の 2 つに分かれる。

1. 常温：温度変化をとくに制御しない。したがって温度は気温と共に変動。
2. 一定温度：恒温槽 (後述) を使って温度を一定に保った。

6.2.2 実験セットアップと実験方法

実験のセットアップを図 6.6 に示す。GPS クロックを LVTTL 矩形波にし、Time Master でそれを源信として Time-Code を生成・送信する過程は前節と同じである。User Node には、5.3 節の SpaceWire FPGA と User FPGA を用いたので、Time Master から Time-Code を受け取ると SDRAM に TI と LocalTime の照合表が書き込まれていく。また、2 枚の DIO board 間の SpaceWire リンクにルーターを挟まなかった。実験 2 では、User Node の DIO board を恒温槽に入れた。

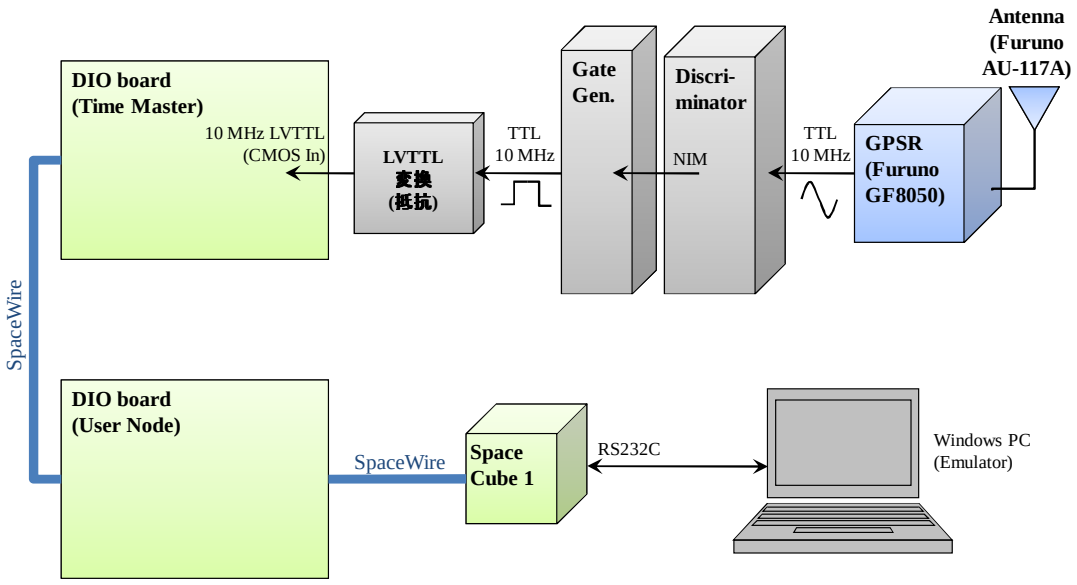

図 6.6 時刻安定度測定実験のセットアップ

恒温槽

恒温槽とは、中の温度を制御できる装置のことである。我々が用いた恒温槽は温度が安定するまで 30 分ほどかかるので、はじめ 30 分間は定値運転で運転し、測定はしなかった。その後は、およそ 1 時間の間、恒温槽の温度計で計測した、恒温槽内の温度安定度は ± ∼0.1 K であった。運転開始から 1 時間半後は、ほぼ安定した温度であった。

6.2.3　結果 1:常温下での時刻安定度

まず、とくに恒温槽を使わず、常温のもとで時刻安定度を 2 回測定した。

1 回目の測定

1 回目の測定時間はおよそ 23 時間 (TI=5.3×10^6 相当) だった。測定開始時刻は午後 11 時、測定終了時刻は午後 10 時だった。また、この日の気温変動は外気で ± ∼5 ℃だった。

図 6.7 は、1 回目の測定を行った結果である。横軸は　TI で差し渡し一日ちかい変動を見ているため、先の節のジッターには影響されないタイムスケールの変動をみた実験結果である。昼ごろもっとも高くなり、未明にもっとも低くなるという一日の気温の変化と同期した LocalTime のゆらぎがみられる。

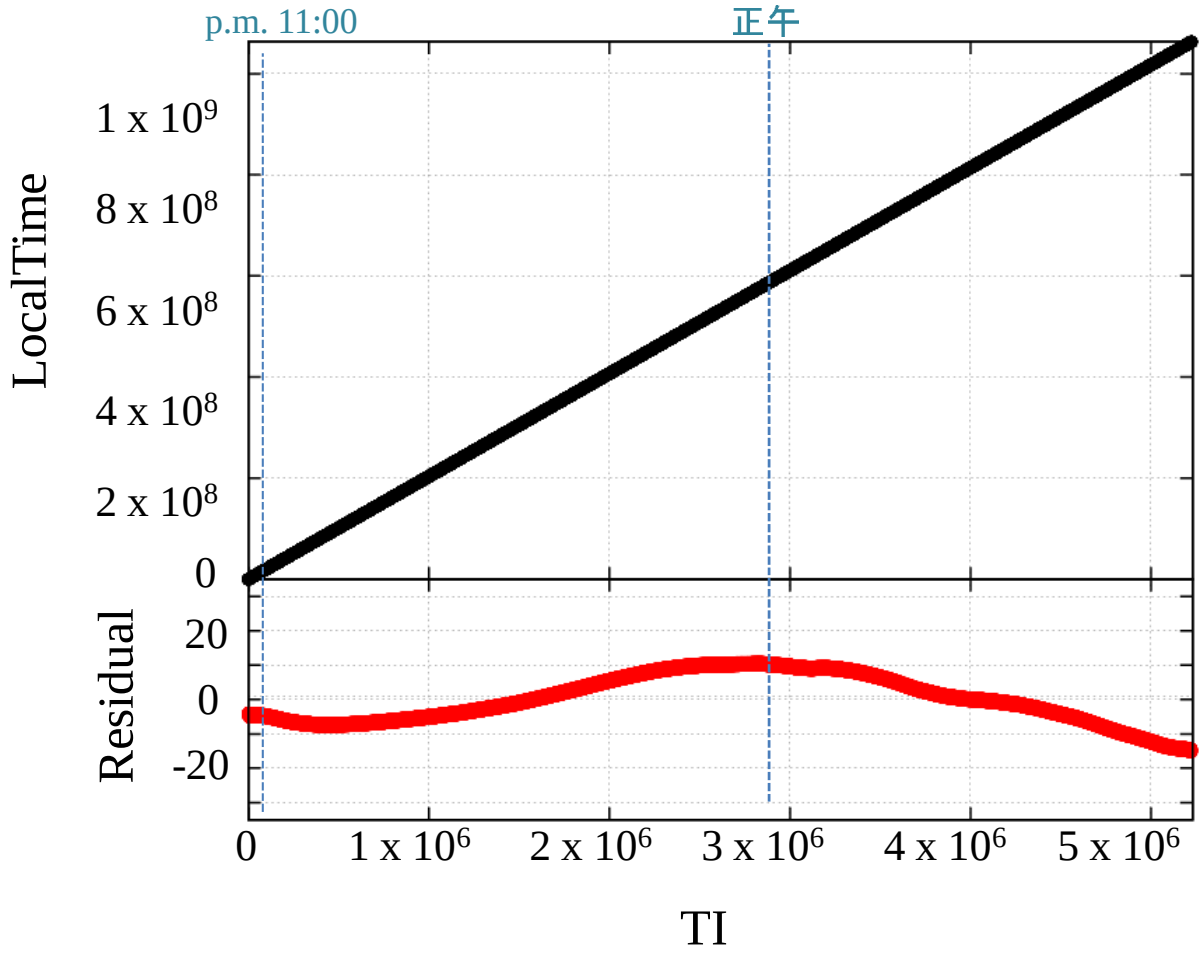

図 6.7　ある日の常温下での時刻安定度。横軸は User Node で合成した TI、縦軸は LocalTime、下段は、データ点を一次関数で fit したときの残差を示している。

2 回目の測定

2 回目の測定時間はおよそ 24 時間 30 分 (TI=5.6×10^6 相当) だった。測定開始時刻は午後 6 時、測定終了時刻は午後 6 時 30 分だった。また、この日の気温変動も外気で ± ～5 ℃だった。

図 6.8 は、2 回目の測定を行った結果である。こちらも一日の気温の変化と同期した残差のゆらぎがみられるが、やや安定度が小さい。

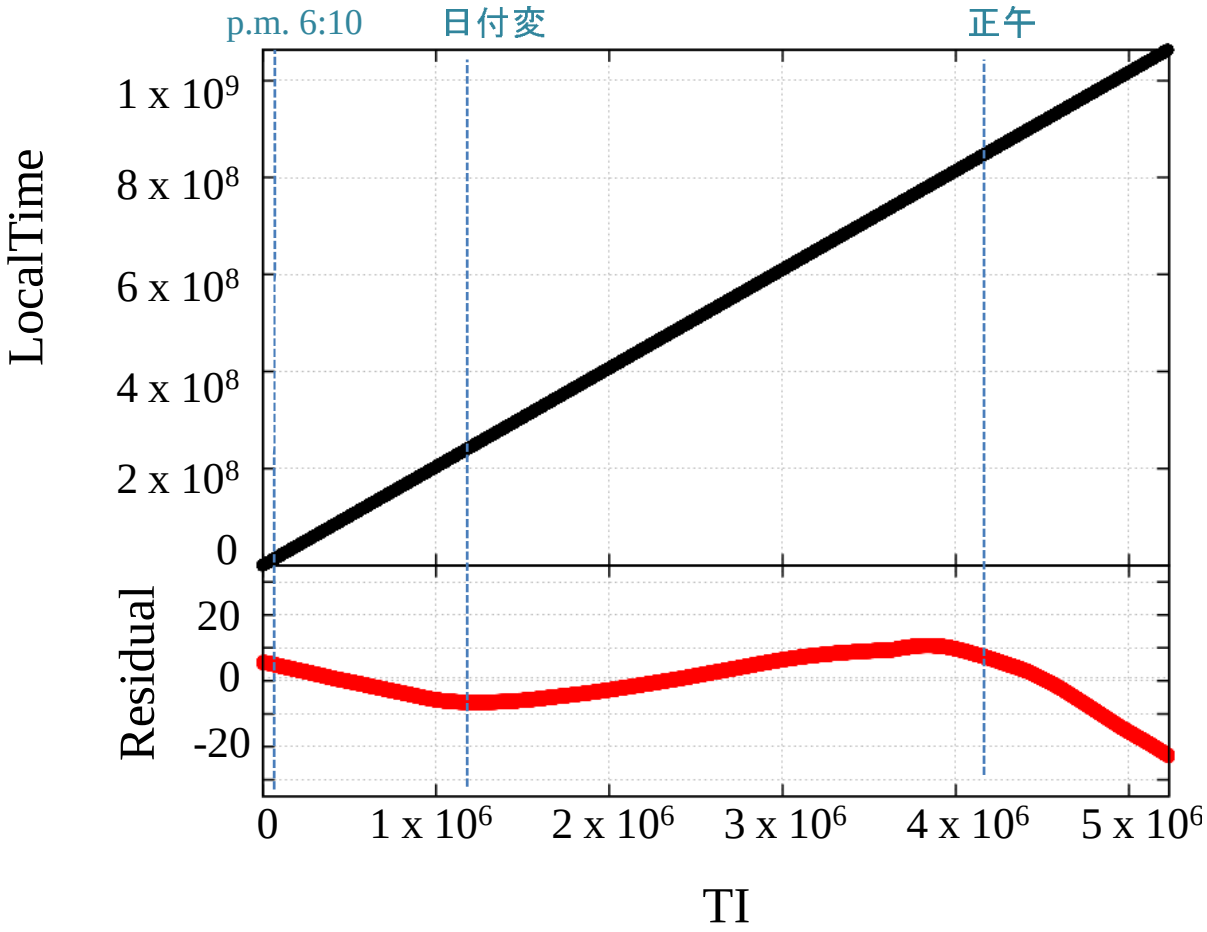

図 6.8　図 6.7 と別の日に測定した常温下での時刻安定度。凡例やラベルは図 6.7 と同じ。

6.2.4　結果 2:一定温度での時刻安定度

我々は、前節で測定した時刻安定度の変化が温度によるものかを確かめるため、温度を一定に保ったときの時刻安定度を測定した。User Node の水晶発信器の温度を一定に保つ必要があるので、User Node のみを恒温槽に入れた。

図 6.9 は、温度 25 ℃で 6 時間、定値運転したときの時刻安定度を示している。温度変動がない場合は、ほぼ時刻は安定しており、LocalTime の分解能 (80 μsec) の以上の安定度であることがわかる。

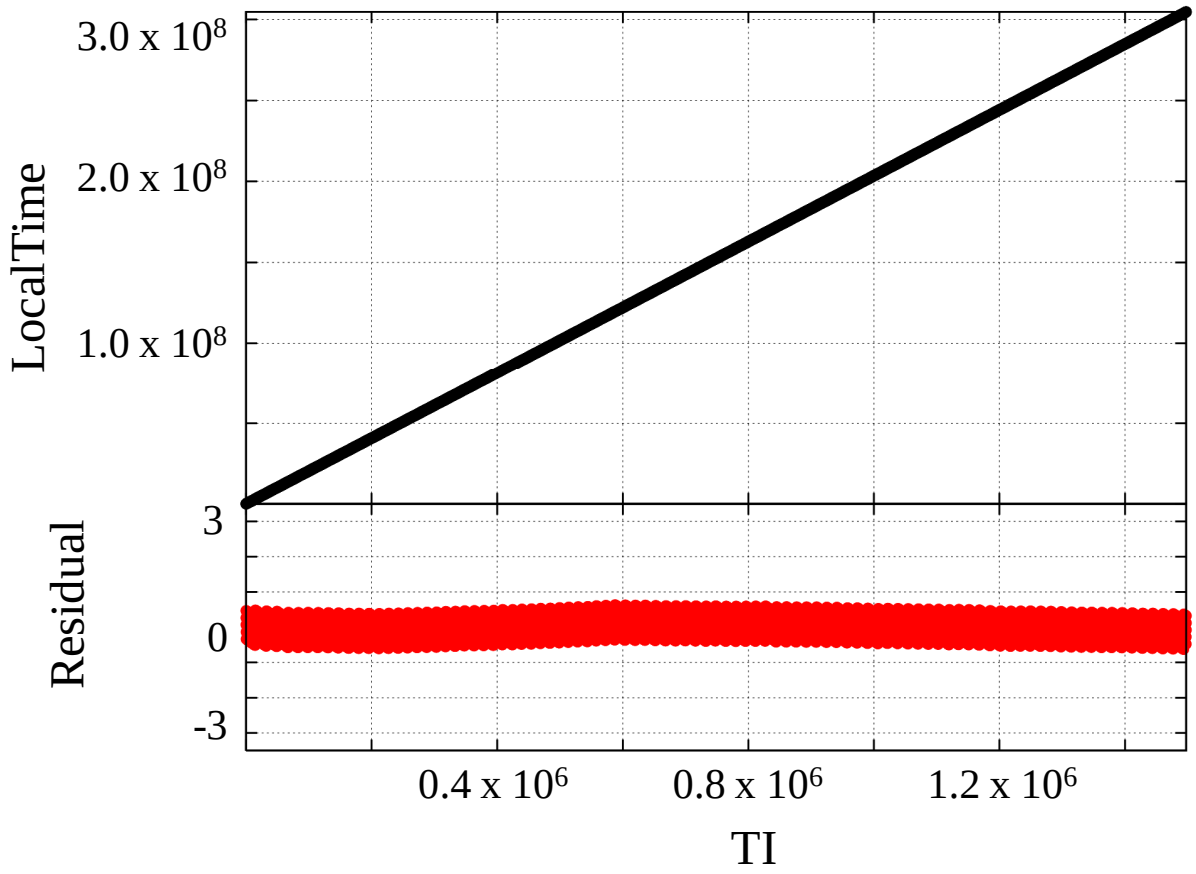

図 6.9　一定温度での時刻安定度。凡例やラベルは図 6.7 と同じ。

6.2.5 考察

結果 1 と結果 2 の比較から、LocalTime の時刻安定度は、温度の変化に影響されることがわかる。これは、水晶発振器の振動周波数が、温度とともに変化するからである。

結果 1 の常温下では、1 回目の測定 (図 6.7) において 23 時間の測定時間に対し、残差のゆらぎ幅およそ LocalTime 30 カウント分なので、

$$30 \times 80\ \mu\text{sec} = 2400\ \mu\text{sec}$$

よって、1 秒あたりの LocalTime のゆらぎ幅 (時刻安定度) は、

$$2400\ \mu\text{sec} \div (23\ \text{hours}) = 2.9 \times 10^{-8}\ \text{sec/sec}$$

となる。同様にして、2 回目の測定 (図 6.8) においては、3.6×10^{-8} sec/sec となる。もし水晶発振器のドリフトの振舞いが ASTRO-H 搭載品と同じならば、要求時刻精度を Y sec、TI と LocalTime のサンプル周期を P sec とすると

$$Y > (3.6 \times 10^{-8}\ \text{sec/sec}) \times P$$

となる P を選べばよい。よって $Y = 30\ \mu$sec から、

$$P < 200\ \text{sec}$$

が見積もれる。次の節では、さらに詳細な測定から、適切な P を見積もるための実験について述べる。

	測定時間 (hours)	**測定時間中の LocalTime ゆらぎ** (μsec)	**時刻安定度** (sec/sec)
1 回目	23	2400	2.9×10^{-8}
2 回目	24.5	3200	3.6×10^{-8}

表 6.2 時刻安定度の測定結果のまとめ

6.3　TI と LocalTime のサンプル周期と時刻安定度の測定

6.3.1　概要

これまで、時刻精度とは Time-Code のジッタと LocalTime の時刻安定度に影響されるので、その両者を実測する実験を行ってきた。特に後者は、時間とともに変動する温度に大きく影響されていた (温度ドリフト)。LocalTime の時刻安定度は、表 6.2 に示す通り、1 秒あたり 40 nsec 程度であった。すなわち、このゆらぎのトレンドを掴みさえすれば、時刻精度に影響するパラメーターは Time-Code のジッタのみになる。

LocalTime の刻みの速度のトレンドを正確に追うためには、TI と LocalTime の照合表 (時刻 HK) をできるだけ細かく記録すればよい (以下、時刻 HK を記録する時間的間隔を「サンプル周期」と呼ぶことにする)。しかし、軌道上では SDRAM の容量は限られているし、SpaceWire リンクで転送できるデータ量の制限の中で他のデータなどを圧迫しないようにしなくてはならない。

我々は、適切なサンプル周期を選ぶため、温度安定度とサンプル周期を変えたときの時刻精度を実測した。この節では、この実験の結果と考察をまとめる。

6.3.2　方法

サンプル周期と時刻精度の相関は、サンプル周期 P (sec) の時間のあいだ時刻がどれだけゆらいだかを調べることでわかる。時刻安定度を D (sec/sec) とすると、ゆらぎは、$D \times P$ となるはずである。より具体的には、まず、時刻 HK(TI と LocalTime の照合データ) をできるだけ細かくとっておく。次に、調べたいサンプル周期ごとに時刻 HK を選び出す。そして、選び出した 2 データ間を一次関数で内挿する。この関数と、細かくとっておいた時刻 HK との残差から、時刻のゆらぎを計算する。

6.3.3　実験セットアップと実験方法

実験セットアップは、前節の図 6.6 と同じである。ただし、User Node は恒温槽に入れて温度変動を制御した。また、今回は Time-Code のジッタの影響も加味して測定するため、2 枚の DIO board の間にルーターを挟んで (1 hop) の測定も行った。

User Node の User FPGA は、以下のように変更した。

サンプル周期

実際に SDRAM に記録するサンプル周期は、Time-Code と同じ 64 Hz = 15.6 msec にした。(あるサンプル周期の間の時刻安定度を測るときは、前述のようにした)

LocalTime の分解能

LocalTime の分解能は、これまで 80 μsec にしてきたが、今回はそれより小さな時刻精度を測るため、DIO board の水晶発振器の周期と同じ 1/(48 MHz) $\sim$ 20 nsec とした。

6.3.4 結果

User Node を恒温槽に入れ、槽内をある一定の温度安定度に保ったときの時刻安定度を測定した。その際、
(1) 温度変化量は ±5 ℃,±10 ℃, ±20 ℃、
(2) 温度は 25 ℃を基準値とし、
(3) 温度変化のタイムスケールは ASTRO-H で予想される周回周期 90 分
とした。なお、見込まれる温度変化はミッション機器ごとに違うが、過去の経験から最悪でも ±20 ℃程度と予想される。また、本節の実験において、ルーターは介さなかった。

図 6.10 は、上記の環境下で測定したサンプル周期と時刻安定度の相関である。サンプル周期が粗くなると時刻安定度が悪くなるのは図 D.3 と同様の結果である。サンプル周期の最大値 80 秒は要求時間分解能で LocalTime が回りきるまでを基準としており、これ以上大きくするのはロジック設計面で適切でない。よって、サンプル周期 80 秒が最悪の時刻安定度となる。一方で、温度変動値が大きくなると、時刻安定度が顕著に悪化するのがわかる。したがって、80 秒のサンプル周期かつ温度安定度 ±20 ℃における時刻安定度が最も最悪のケースである。

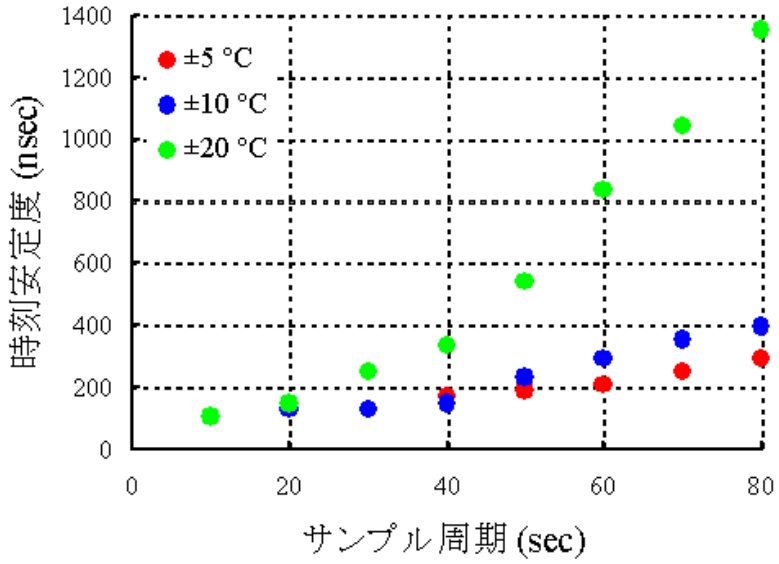

図 6.10 LocalTime の時刻安定度と温度変動値の相関。恒温槽内の温度変動値については、赤が ±5 ℃, 青が ±10 ℃, ±20 ℃を示している。縦軸、横軸に関しては、図 D.3 と同じ。

6.3.5 考察

適切なサンプル周期

本実験から、適切なサンプル周期を定めることで DIO board の水晶発振器のドリフトの影響を小さくできることがわかった。ASTRO-H の場合、Time-Code のジッタは 6.1 節で触れたとおり、1 hop ごとに 350 μsec だった。例えば、ドリフトの影響をこれより小さい範囲で収めるならば、サンプル周期を 10 sec 以内にすればよい。

要求を満足するか

結果 2 でも示したように、80 秒のサンプル周期かつ温度安定度 ± ∼20 ℃における時刻安定度 1.3μsec が最も最悪のケースである。これは Time-Code のジッタ 4.4 μsec と合わせても、要求時刻精度 30 μsec より良い。

6.4 本章の結論

ASTRO-H での要求時刻精度を達成できるか

本章で述べた実験から、時刻 tick 内挿法における時刻精度 Δt は、Time-Code のジッタ σ_{tc}(sec) と、LocalTime の時刻安定度 D(sec/sec)、TI と LocalTime のサンプル周期 P(sec) その他のジッタ σ_{other}(sec) から決まる。

$$\Delta t = \sigma_{tc} + \sigma_{other} + D \times P$$

この式に実測値を代入 (σ_{tc} のみ ASTRO-H 相当のものに置き換え) すると、

$$\Delta t = 1.8\ \mu\text{sec} + 4 \times 10^{-8}\ \text{sec/sec} \times P$$

ここで、σ_{other} は非常に小さい (20 nsec) ので無視した。Δt を ASTRO-H に必要な時刻精度は 30 μsec とすると、$P = 600$ sec となる。原理的には、ここまで遅いサンプル周期でも要求時刻精度を満たせるはずだが、LocalTime が回りきるまでに設定しなければならないので、最大でも $P = 80$ sec 程度にすべきである。また、P =1 sec で十分な時刻精度を達成できることが確認できているが、このサンプル周期ならば他のデータなどを圧迫しない。

衛星搭載品の水晶発振器と DIO board のとでは D が違うが、最大でも $\pm$ 20 ℃程度になる見込みである。この場合における、最悪の時刻安定度は 1.3 μsec であり、Time-Code のジッタと合わせても要求時刻精度 30 μsec を満たす。すなわち、我々の提案した 非同期クロックを用いた時刻 tick 内挿法は、P がいくつであっても (ただし $P < 80$ sec)、かならず ASTRO-H の要求時刻精度を満たす。

第 7 章

まとめと今後

パルサーなどの観測をするため、ASTRO-H 衛星に必要な時間分解能は 10 μsec、時刻精度は 30 μsec 程度である。しかし、ASTRO-H 衛星で採用予定の SpaceWire を用いて配信される衛星時刻 (TI) では、観測機器に必要な時刻精度を達成できない。そこで、我々は、高い時刻精度を得るための方法として、ASTRO-H のネットワークの末端機器にまで実装することができる、非同期クロックを用いた時刻 tick 内挿法を提案した。これは、観測機器の FPGA 搭載ボードで TI と別に自身の水晶発振器に同期した細かい刻みの時刻カウンタ (LocalTime) をもち、TI と LocalTime を照合することで高い時刻精度を得る方法である。

さらに、我々は、自ら提案した非同期クロックを用いた時刻 tick 内挿法を実際に SpaceWire 搭載機器に実装して、時刻精度を測定する実験を行った。時刻精度は、Time-Code のジッタ、および TI と LocalTime の照合データをとるサンプル周期と LocalTime の時刻安定度によって決まる。実験結果から、Time-Code のジッタは、ASTRO-H で採用されるネットワークや機器でおよそ 4.4 μsec 以下であることがわかった。さらに、サンプル周期は LocalTime が回りきる 80 sec 以内にすれば、要求時刻精度が達成できることが見積もれた。以上から、我々の考案した方法で、ASTRO-H の観測機器に必要な時刻精度を達成できることがわかった。

今後は、以下のような作業が必要である。

衛星搭載品での実験

まず、我々の採用した 非同期クロックを用いた時刻 tick 内挿法が ASTRO-H 衛星の観測機器に搭載でき、実際に必要な時刻精度が得られることを確かめるため、ブレッド・ボード・モデル (BBM) 品での実験が必須である。今後、我々は、MIO board を使った同様の実験を行っていく予定である。また、実際に、地上で衛星搭載品で衛星全体のネットワークを構築して、時刻配信や時刻付けの試験を行うことも必要である。

GPS 衛星の捕捉状況と時刻精度

本論文で我々が下した結論には、「SMU で生成される TI(TIME DATA + Time-Code) の絶対時刻精度は無限小」という仮定があった。しかし、これは GPS 衛星を十分な数だけ捕捉している場合に限られる。

ASTRO-H で採用予定の GPSR は、GPS 衛星を 1 機も捕捉していない時間帯の前後数分間はクロックを出さない仕組みになっており [25]、この間は SMU の内部クロックと同期した衛星時刻 (TIME DATA, Time-Code) を配信することになっている。SMU の内部クロックは、当然ながら、温度変動でドリフト (6.2 節) を起こすので、時刻精度は GPS 捕捉時より悪くなることが懸念される。

GPS 衛星を捕捉しているかいないかは GPSR が出すテレメトリにあるフラグから判断できる。こ

れを SMU で参照できるようにし、衛星時刻の精度が信頼できないことをテレメトリに出す必要がある。また、GPS 衛星を捕捉していない場合の衛星時刻の精度を推定するため、SMU の水晶発振器に温度計を取り付けて温度ドリフトの様子をテレメトリ出力するなどの方法が要求される。このためには、SpaceCube2 を使った地上実験で、あらかじめ SMU の水晶発振器の温度ドリフトと衛星時刻精度の相関を実測しなくてはならない。また、GPS 衛星の可視解析や、GPS 衛星の捕捉数と絶対時刻精度の関係などを明らかにしていくことも必要である。

付録 A

水晶発振器のドリフトの確認実験

TI と LocalTime を照合する 非同期クロックを用いた時刻 tick 内挿法 (第 4 章や第 6 章を参照) では、時刻精度が LocalTime の時刻安定度に依存する。温度変動に対して、時刻安定度がどう変わるか、は第 6 章で議論してきた。

「温度変化」という現象は、3 つのパラメータで表現できる。**(1) 温度の変化分はいくらか。(2) 何度を基準に変化したのか。(3) どのくらいのタイムスケールで変化を起こしたか。** 本実験では、とくに (2) をクローズアップした実験を行った。

A.1　実験セットアップと方法

6.2.2 節に同じ。恒温槽は、プログラム運転 (槽内の温度変化の様子を設定して運転) を行った。この温度変動の様子を A.1 に示す。

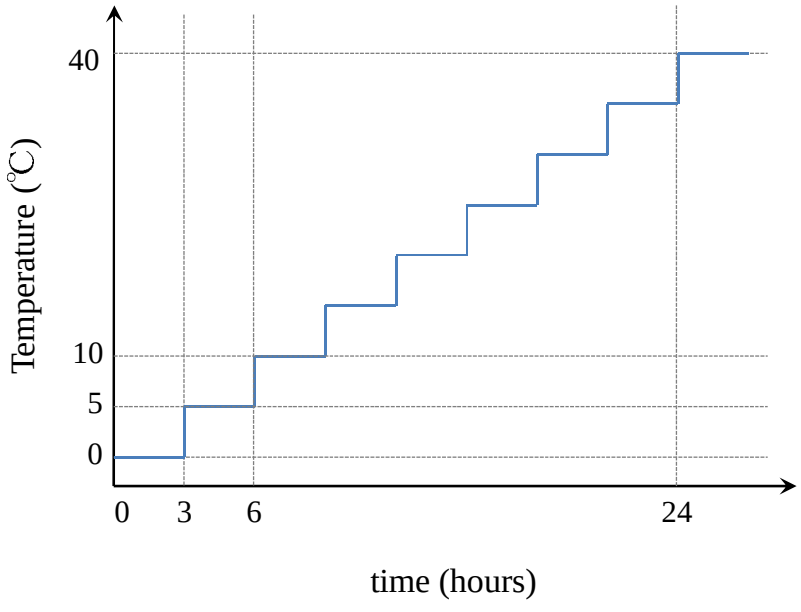

図 A.1　階段状温度変動での実験における恒温槽内の温度変動

A.2　結果

図 A.2 は、図 A.1 の温度変動下で測定した TI と LocalTime のトレンドである。温度が変化した時間に残差の振る舞いも変化していた。また、残差の傾きは低温の方が急になっていた。

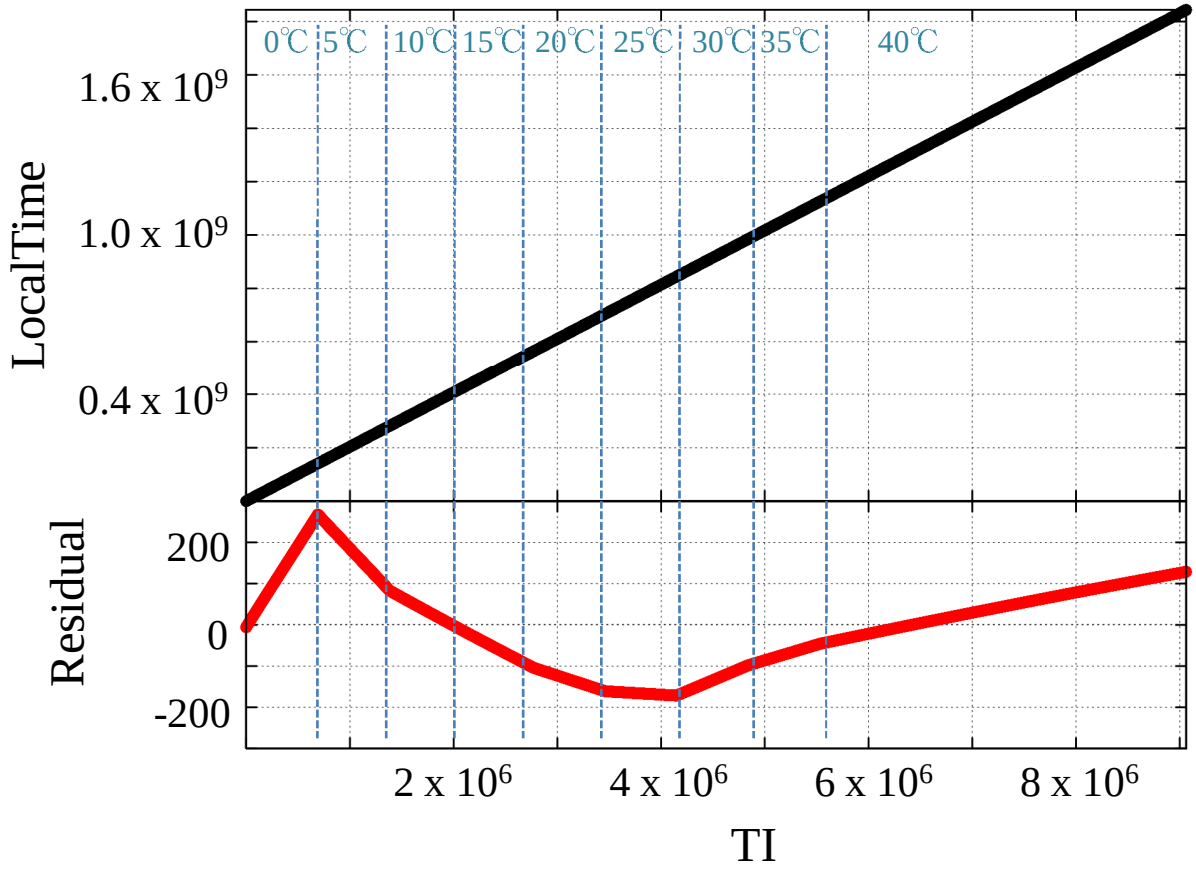

図 A.2　図 A.1 に示す温度変動下での時刻安定度。凡例やラベルは図 6.7 と同じ。

A.3　考察

前節の結果から、低温のほうが、同じ温度変動をしても、水晶発振器の周波数が大きく変わることがわかった。すなわち、温度変動のパラメータのひとつである「基準温度」が LocalTime の時刻安定度に影響する。よって、高い時刻安定度を得るためには、できるだけ低温を避け、温度の変化分が少ない環境に MIO board を配置することが望ましい。

付録 B

イベント時刻付け機能の確認

実際にイベントに時刻付けをする機能の実装は、第 5 章で述べた。この機能を使った実験については [26] に詳細が書かれる予定なので、ここではおおまかな機能について確認した実験を載せる。

B.1 セットアップ

実験のセットアップを図 B.1 に示す。イベント信号の代わりに、Gate Generator からの信号を利用した。Trigger Switch を押すと、LVTTL 信号が User Node に入力される。

User Node の SDRAM に記録されたイベントの時刻データ (LocalTime) から、時刻 HK データによる内挿 (第 4 章参照) で TI 相当の時刻を求めた。内挿を行うための、Perl スクリプトを作成した。スクリプトの詳細は B.3 節を参照されたい。

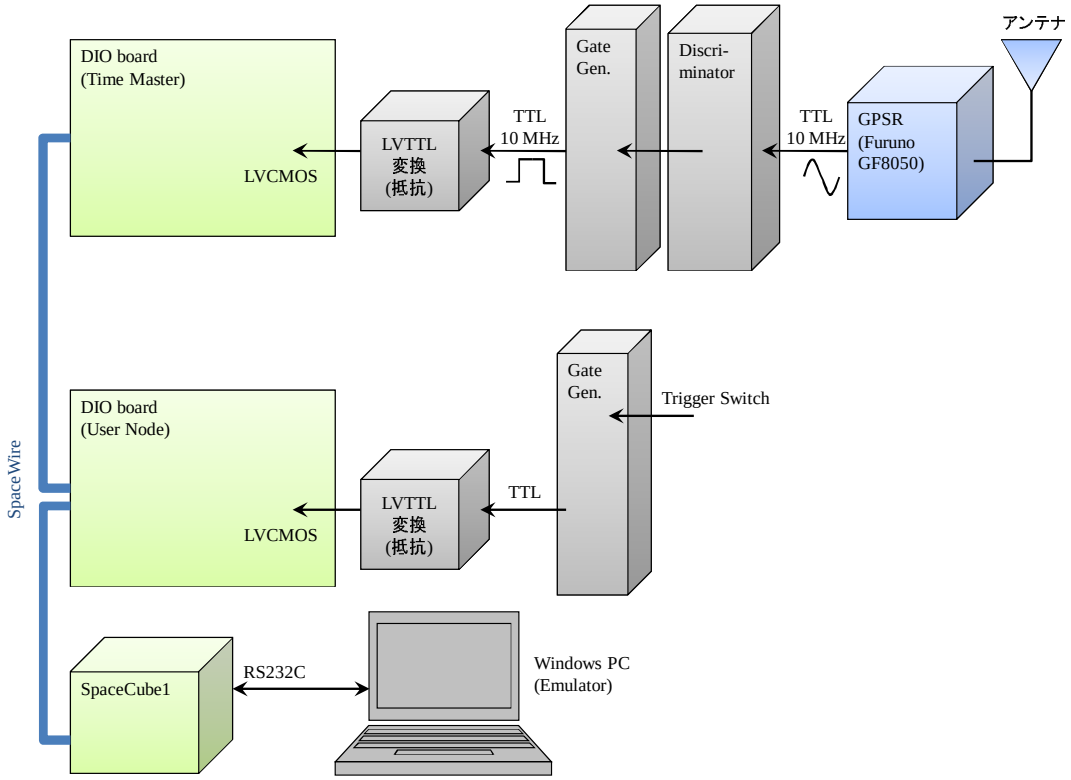

図 B.1　イベント時刻付け機能の確認実験セットアップ

B.2 結果

LocalTime の時間分解能は 80 μsec とした。およそ 5 秒おきに手で Trigger Switch を 3 回押した。結果を表 B.1 に示す。TI は 1 sec で 64 増えるので、データは 5 秒間隔でとれていたことがわかる。

イベントの LocalTime	推定した TI 相当時刻	前の TI (時刻 HK)	前の LocalTime (時刻 HK)	後の TI (時刻 HK)	後の LocalTime (時刻 HK)
655151	352.2236387374	320	648595	384	661616
724719	694.1648106904	640	713699	704	726720
784839	989.6677674526	960	778803	1024	791824

表 B.1 イベントの時刻と前後の時刻 HK データ

B.3 イベント時刻付けツールのソース・コード

```
#!/usr/bin/perl

#EVTTIME.pl
#==========
#2009.11.06 - ver 1

if (@ARGV != 3){
    print "Usage : \n";
    print "./evttime.pl IN_HK.dat IN_EVT.dat OUT.dat\n";
    print "----------\n";
    print "IN_HK.dat  = HK Data file (must be in dec)\n";
    print "IN_EVT.dat = EVT list file (must be in dec)\n";
    print "OUT.dat = EVT time table Output\n";
    exit(0);
}

# display the Title
print "evttime.pl running...\n";

# arguments
$hk  = $ARGV[0]; # HK data
$evt = $ARGV[1]; # EVT data
$out = $ARGV[2]; # Output data

open (EVT, "$evt") || die "can not open $evt.";
open (FOUT, "+>$out") || die "can not open $out";

#-----------------------------
# make HK & EVT data in dec.
#-----------------------------

$hk_ti_p  = hex(ffffffff);
$hk_lt_p  = hex(ffffffff);

while(<EVT>){
    if(/(\S+)/){
  $evt_lt = $1;

  open (HK, "$hk") || die "can not open $hk.";

  while(<HK>){
```

```
42        if(/(\S+)\s+(\S+)/){
43      $hk_ti = $1;
44      $hk_lt = $2;
45
46      if ($hk_lt > $evt_lt && $hk_lt_p < $evt_lt ){
47
48          # assume linear function : y=ax+b
49          $a      = ($hk_lt - $hk_lt_p)/($hk_ti - $hk_ti_p);
50          $b      = $hk_lt_p - $a * $hk_ti_p;
51          $evt_ti = ($evt_lt - $b)/$a; # x=(y-b)/a
52
53          printf ("Event LocalTime : %6.1f\n", $evt_lt);
54          printf ("calculated Event TI : %6.10f\n", $evt_ti);
55          print "Used HK data\n";
56          print "-----------------------------------\n";
57          print "HK   LocalTime      TI\n";
58          print "-----------------------------------\n";
59          print "Post  $hk_lt      $hk_ti\n";
60          print "Pre   $hk_lt_p      $hk_ti_p\n";
61          print "-----------------------------------\n";
62          printf FOUT ("%6.10f %6.10f\n", $evt_lt, $evt_ti);
63      }
64
65      #present value -> previous value
66      $hk_ti_p  = $hk_ti;
67      $hk_lt_p  = $hk_lt;
68        }
69    }
70    close(HK);
71      }
72  }
73
74  close EVT;
75  close FOUT;
```

付録 C

Time Master の User FPGA の VHDL コード

Time Master の User FPGA の VHDL コードを一部掲載する。

```
--UserFPGA_DIO.vhd
--Time-Code 64HzSend Module

------------------------------------------------------
--Declarations of Libraries used in this UserModule
------------------------------------------------------
library ieee,work;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library unisim;
use unisim.Vcomponents.ALL;

------------------------------------------------------
--Entity Declaration of UserFPGA
------------------------------------------------------
entity UserFPGA_DIO is
  port (
    Clock        : in std_logic;
    GlobalReset    : in std_logic;
    CMOSIn       : in std_logic_vector (7 downto 0);
    CMOSOut      : out std_logic_vector (7 downto 0);
    LEDs         : out std_logic_vector (1 downto 0);
    --timecode related-signals
    TickIn     : out std_logic;
    TimeIn     : out std_logic_vector(5 downto 0);
    Revision       : in std_logic_vector (15 downto 0)
  );
end UserFPGA_DIO;
```

```
31
32
33  ----------------------------------------------------------
34  --Behavioral description of this UserModule
35  ----------------------------------------------------------
36  architecture Behavioral of UserFPGA_DIO is
37
38    -- LED output
39    signal led : std_logic_vector (1 downto 0);
40
41
42  ----------------------------------------------------------------
43  -- GPS Clock no syuhasu to
44  -- TimeCode no syuhasu wo nyuryoku shite kudasai
45  ----------------------------------------------------------------
46    constant GPSClock_Hz : Integer := 10000000; -- in Hz
47    constant TimeCode_Hz : Integer := 64; -- in Hz
48
49  -- GPSClock
50    signal GPSClock : std_Logic := '0';
51    signal GPSClock_previous : std_Logic := '0'; -- 1 clock mae no atai
52    signal ClockCounter : integer :=0;
53    signal TimeCodeCounter : std_logic_vector(5 downto 0) := (others => '0');
54    signal TickIn_sync : std_logic := '0';
55    signal TickIn_sync_previous : std_logic := '0';
56    constant count_max : Integer := GPSClock_Hz / TimeCode_Hz;
57
58
59    ----------------------------------------------------------
60    --Beginning of behavioral description
61    ----------------------------------------------------------
62    begin
63
64    --static relations
65    GPSClock <= CMOSIn(7);
66    LEDs <= led;
67
68
69    process(GPSClock,GlobalReset)
70    begin
71      if (GlobalReset='0') then
72        TickIn <= '0';
73        TimeIn <= (others => '0');
74        led(1) <= '0';
```

```
75      elsif (GPSClock = '1' and GPSClock'Event) then
76        if (ClockCounter=count_max-1) then
77          if (TimeCodeCounter=TimeCode_Hz-1) then
78            TimeCodeCounter <= (others => '0');
79            led(1) <= not led(1);
80          else
81            TimeCodeCounter <= TimeCodeCounter + 1;
82          end if;
83          TickIn <= '1';
84          TimeIn <= TimeCodeCounter;
85          ClockCounter <= 0;
86          led(0) <= not led(0);
87        else
88          ClockCounter <= ClockCounter + 1;
89          TickIn_sync <= '0';
90        end if;
91      end if;
92
93    end process;
94
95
96  end Behavioral;
```

付録 D

TI と LocalTime のサンプル周期と時刻安定度の測定: 常温下・ルーターあり

D.1 概要

6.3 節 (50 ページ) の実験と同様ながら、User Node を恒温槽に入れず常温のままで時刻安定度を測定した。したがって、温度変動は制御していない。

D.2 実験セットアップと実験方法

実験セットアップや方法は、前節の図 6.3 節と同じである。ただし、常温下で実験を行うため、User Node は恒温槽に入れなかった。また、今回は Time-Code のジッタの影響も加味して測定するため、2 枚の DIO board の間にルーターを挟んで (1 hop) の測定も行った。

D.3 結果

まず、hop なし (ルーターなし) の測定を行った。そのデータをもとに、あるサンプル周期の間における時刻安定度を調べた。サンプル周期は、1/64 sec, 1 sec, 10 sec, 30 sec, 60 sec について調べた。1/64 sec のときは、水晶発振器の温度ドリフトの影響は無視でき、ジッタ (90 nsec : 6.1 節) と同じ値となった。1 sec より大きなサンプル周期では、図 D.1 のようにドリフトの影響がみられた。1 sec のサンプル周期ではおよそジッタに上乗せして <40 nsec のゆらぎがあった。これは、前節の結果と合っている。

次に、1 hop(ルーター 1 個を挟む) の測定を行った。サンプル周期 1 sec, 10 sec, 30 sec, 60 sec について、図 D.1 のような結果が得られた。no hop より残差が大きいが、そのトレンドから DIO board の水晶発振器のドリフトと違ったゆらぎがみられる。

サンプル周期と、その間の時刻安定度の相関をとると、図 D.3 のようになる。no hop、1 hop ともに、サンプル周期が大きくなると時刻安定度も悪くなっている。

D.4 考察

1 hop で時刻安定度が悪化する原因

1 hop でサンプル周期 60 sec のとき、1.8 μsec 程度の時刻安定度で、no hop より 5 倍以上悪い。もし、1 hop ごとに 5 倍時刻精度が悪化すると、ASTRO-H の場合、Time Master (SMU) から User Node (SpaceCard) まで 4〜5 個のルーターを介するので、300 $ns \times 5^4 \sim 5^5 = 200 \sim 1000$ μsec となるので、要求時刻精度 (30 μsec) を満たせない。この結果は、以下のような原因が考えられる。

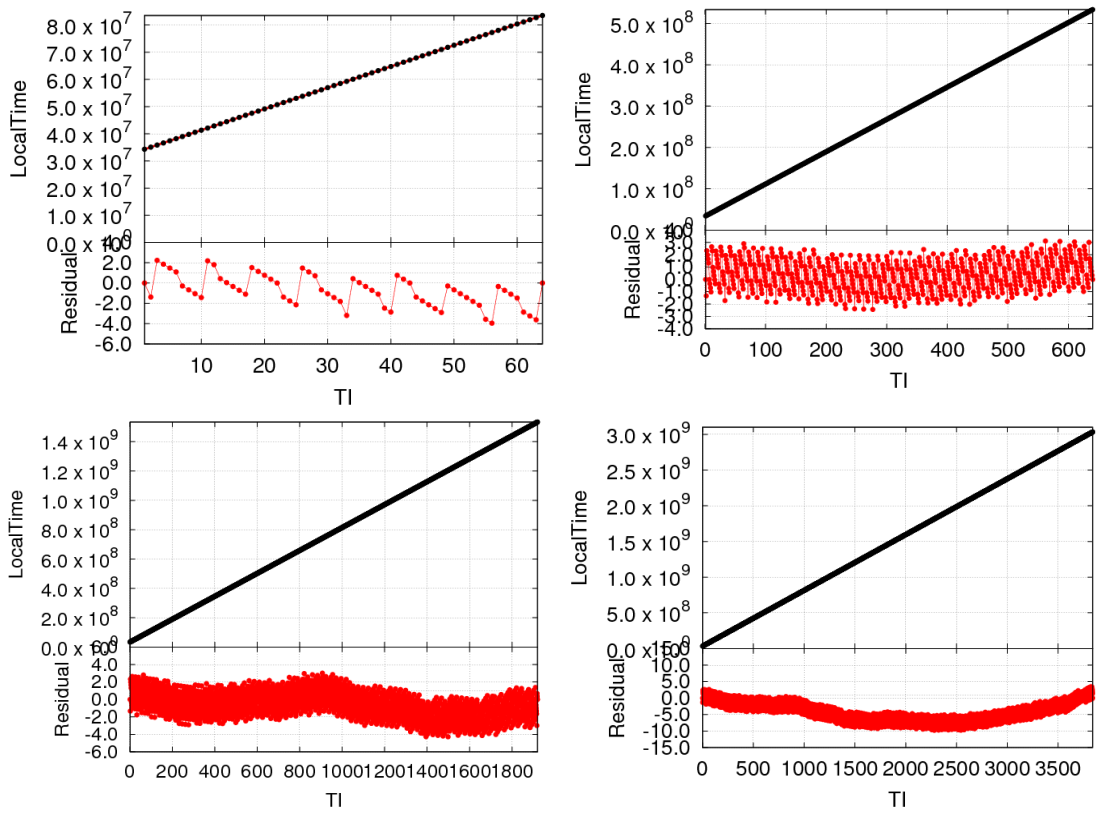

図 D.1 hop なしにおけるサンプル周期と時刻安定度。サンプル周期は、(a) 1 sec, (b) 10 sec, (c) 30 sec (d) 60 sec

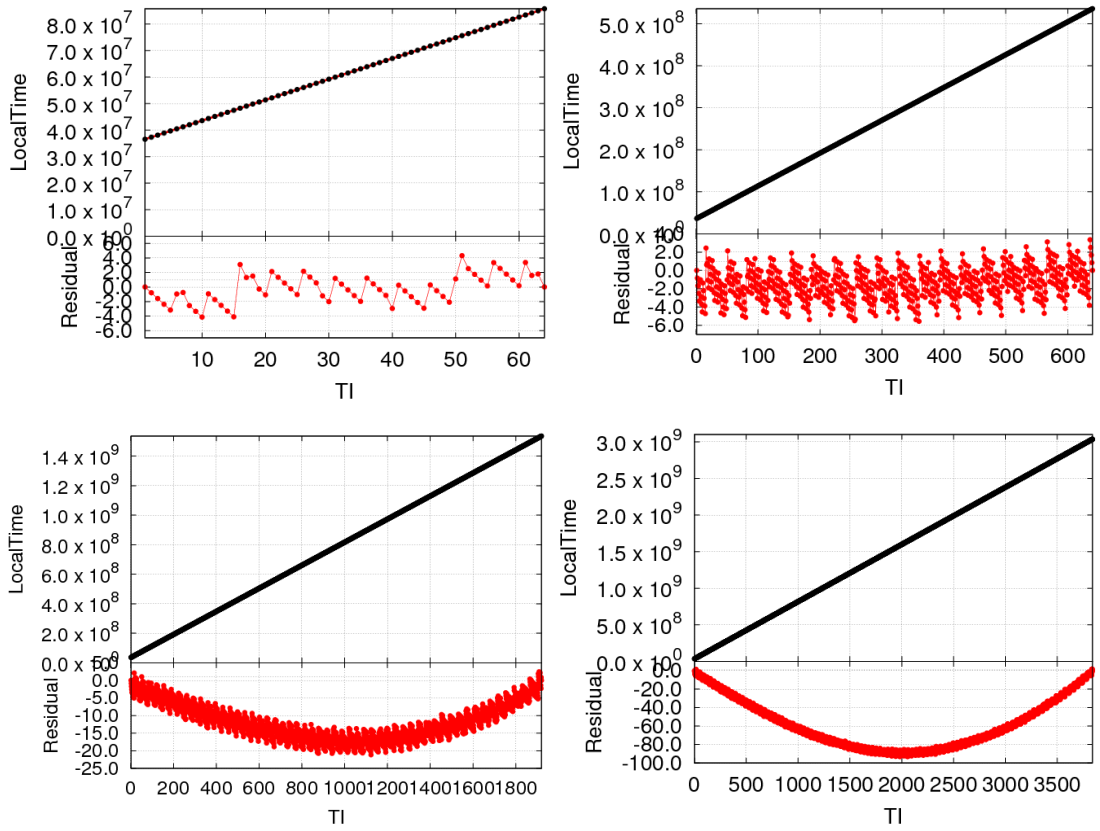

図 D.2 1 hop におけるサンプル周期と時刻安定度。サンプル周期は、(a) 1 sec, (b) 10 sec, (c) 30 sec (d) 60 sec

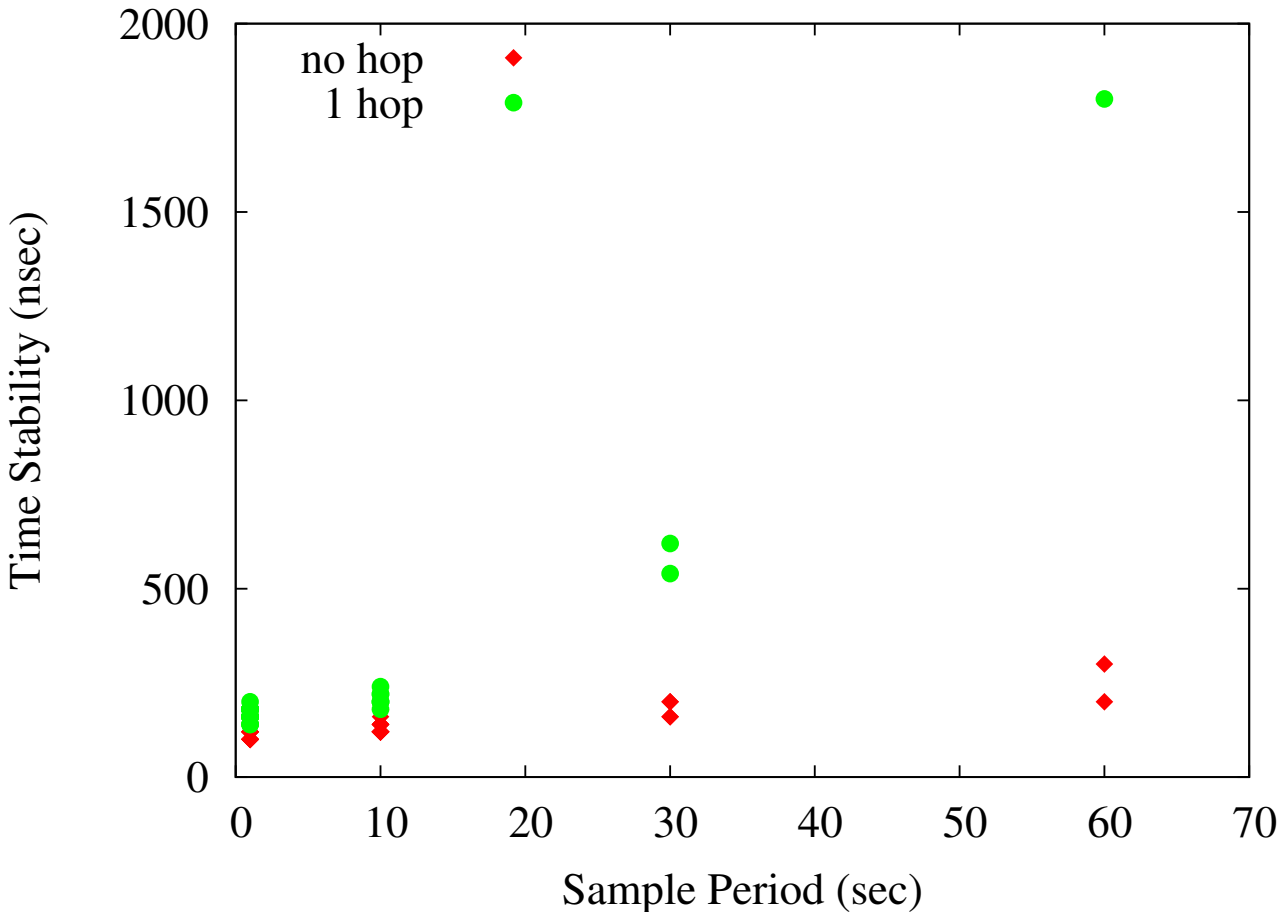

図 D.3　サンプル周期とその間の時刻安定度の相関。同じサンプル周期でプロットが重なっているのは、一回の測定でとれるサンプルが複数あったため

1. たまたま DIO board の温度の変動が激しかった
2. ルーターのクロックと DIO board のクロックの時刻安定度が違っていた

前者は、測定回数を増やすことで明らかになる。後者の場合は、DIO board とルーター両者を温度安定させることで調査できそうであるが、ルーターに使っている SpaceCube 1 は、通電後に急激な温度上昇を起こすことがあるため、測定が困難である。

参考文献

[1] Mitsuda, K., et al. 2007, PASJ, 59, S1

[2] Takahashi, T., et al. 2007, PASJ, 59, S35

[3] Kokubun, M., et al. 2007, PASJ, 59, S53

[4] Terada, Y., et al. 2008, PASJ, 60, S25

[5] Manchester, R. N., Hobbs, G. B., Teoh, A., Hobbs, M., 2005, The Astronomical Journal, 129, 1993

[6] Takahashi, T., et al., 2008, Proc. SPIE, 7011, 14

[7] Mitsuda, K. 2009, High Resolution X-ray Spectroscopy: Towards IXO, p.28

[8] Takagi, S.-I., et al. 2007, Nuclear Instruments and Methods in Physics Research A, 582, 546

[9] 澤田真理, 「次期X線天文衛星 Astro-H 搭載 CCD カメラ SXI の軌道上バックグラウンドの評価およびカメラボディの設計」, 修士論文, 京都大学, 2009

[10] Kokubun, M., et al. 2008, Proc. SPIE, 7011, 21

[11] Tajima, H., et al. 2005, IEEE Transactions on Nuclear Science, 52, 2749

[12] The Fermi LAT collaboration, & Abdo, A. A. 2009, arXiv:0911.2412

[13] Parkes, S. M., et al., “SpaceWire - Links, nodes, routers and networks”, European Cooperation For Space Standardization, Standard No. ECSS-E-50-12A, Issue 1, 24 January 2003

[14] Parkes, S. M., “The Operation and Uses of the SpaceWire Time-Code”, International SpaceWire Seminar (ISWS 2003), ESTEC Noordwijk, The Netherland, 4-5 November 2003 http://spacewire.esa.int/content/TechPapers/documents/SpaceWireTime-CodesISWS2003.pdf

[15] Parkes, S. M., et al., “RMAP Protocol Specification”, European Cooperation For Space Standardization, ECSS-E-50-11 Draft F, 4th December 2006

[16]「ASTRO-H SpaceWire ネットワーク ユーザーズマニュアル」, ASTH-112 Draft, 4th June 2009

[17] 宇宙科学研究本部, 「衛星の機能モデル (FMS)」, GSTOS-201-0.7

[18] 宇宙科学研究本部, 「衛星監視制御プロトコル (SMCP)」, GSTOS-200-0.10, 2009 年 9 月 10 日

[19] Consultative Communitee for Space Data System, “Space Packet Protocol”, CCSDS 133.0-B-1, September 2003

[20] 宇宙航空研究開発機構 誘導・制御グループ, 「次世代衛星搭載用 GPS 受信機の仕様と開発計画」, 平成 21 年 1 月

[21] 「ASTRO-H テレメトリ/コマンド設計基準」, ASTH-111 Rev. 1, 10th November 2009

[22] 湯浅孝行, 日本スペースワイヤーユーザー会, 「SpaceWire/RMAP Library」, 2008

[23] Yuasa T., et al. 2008, “A Portable SpaceWire/RMAP Class Library for Scientific Detector Read Out System”, proceeding of International SpaceWire Conference Nara 2008, pp.173 http://2008.spacewire-conference.org/downloads/Papers/Onboard%20Equipment%20&%20Software/Yuasa.pdf

[24] Yuasa, T., “Development of SpaceWire-based waveform-sampling pulse height analyzer and its

application to a hard X-ray detector", 修士論文, 東京大学, 2008

[25] 「ASTRO-H SMU/GPS 時刻 I/F 概要メモ」, ASTH-NT-D09095, 2009 年 12 月 9 日

[26] 岩瀬かほり, 「次世代X線天文衛星 ASTRO-H における時刻付け方法の検証」, 卒業論文, 埼玉大学, 2010

謝辞

本修士論文を書くにあたり、本当にたくさんの方々にお世話になりました。この場を借りて、心から感謝申し上げます。

指導教官である埼玉大学 寺田幸功 准教授には、現在運用中の「すざく」衛星で実際に使われている時刻付けの方法やノウハウを 1 から教えて頂いたほか、当実験の進め方や FPGA ロジックの組み方、本論文の校正など、本当に様々なことを相談させて頂きました。当実験はもちろんのこと、「すざく」衛星のデータ解析やパルサーの知識など、さまざまなことを教えて頂いております。

東京大学 湯浅孝行さんには、SpaceWire の何から何まで教えて頂きましたが、それ以前に、ログノートの取り方、電源の入れ方までも教わりました。その後もアドバイスやご助力をたくさん賜りました。特に Time-Code のジッタ測定では、湯浅さんの先行実験を参考にさせて頂きました。

埼玉大学 田代信 教授には、実験のノウハウやセンスを教えて頂きました。実験に対するアドバイスもたくさん頂きました。Time-Code のジッタ測定では、田代先生の作成されたスクリプトを使用させて頂きました。

埼玉大学 田代・寺田研究室の学生のみなさまには毎日お世話になっております。岩瀬かほりさんには、理学部の卒業研究にも関わらず VHDL を書くという大仕事をして頂きました。また、本修士論文に掲載した実験の大半をともに行ってくださいました。SXS/PSP チームの瀬田裕美さん、下田優弥くん、朝比奈正人くんとは、SpaceWire に関する情報を共有しながら実験を進めることができました。

大阪大学 能町正治 教授にもアドバイスを頂きました。私が参加させて頂いている ASTRO-H HXI・SGD チームのみなさま、特に宇宙科学研究本部 高橋忠幸 教授、国分紀秀 准教授、東京大学 中澤知洋 講師にもお世話になりました。ASTRO-H SXS/PSP チームのみなさま、特に首都大学東京 石崎欣尚 准教授にもアドバイスやコメントなどを頂きました。

東京大学 牧島・中澤研究室には、実験でたびたびお邪魔させていただきました。埼玉大学 井上研究室のみなさま、井上直也 教授には、実験機器も貸していただきました。理化学研究所 牧島宇宙放射線研究室のみなさま、特に玉川徹 専任研究員と岩橋孝典くんには、実験に必要な機器をお貸し頂きました。