新衛星通信規格SpaceWireを用いたデータ収集システムの開発

広島大学 理学部 物理科学科

B044139

松岡正之

高エネルギー宇宙・素粒子実験研究室

主査： 深澤泰司　副査： 堀 利匡

2008年2月8日

概 要

これまで科学衛星における観測機器、通信機器、各種処理装置間の情報伝達にはハードウェアに合わせた通信プロトコルを開発していた。こうすることでそのハードに最適なプロトコルを作ることができるのだが、通信をする際には相手のプロトコルに合わせた処理をする必要があり、そのための装置の開発をしなくてはならなくなる。これにより時間、コストがかかり、更に異なるプロトコルを通しているため、信頼性を確保するのに全てのプロトコルを考慮しなくてはならなくなる。これを解消するため、衛星の統一通信規格として開発されたのが SpaceWire である。

本研究では SpaceWire を用いたデータ収集システムの基礎を行った。Analog-Digital Converter(ADC) ボードを自作し、アナログ信号を ADC ボードに入力して変換されたデジタル信号を SpaceWire 搭載 Digital I/O ボードを用いて信号を処理する。処理は Digital I/O ボードに搭載されている LSI、FPGA にハードウェア記述言語の VHDL で目的の処理を行う回路を実装して実現した。データ処理を行ったあとのデータは、SpaceWire の開発プラットホームである SpaceCube を通じて PC に転送し、そのデータが解析可能かどうか考察する。そして、将来 SpaceWire を用いて衛星搭載半導体検出器の多チャンネル同時読み出しシステムを開発するための基礎を習得した。

目 次

図 目 次

第1章　序論

1.1　背景

宇宙では我々の想像もつかないような現象が多くある。これまでに超新星残骸や銀河団、ブラックホールなどの高エネルギー天体からは硬X線、ガンマ線領域のエネルギーを持つ放射が観測されている。これらは逆コンプトン散乱、シンクロトロン放射、制動放射など、相対論的エネルギーまで加速された粒子からの放射であることが分かってきた。これらはいずれも非熱的な過程である。我々の住む環境では粒子が十分に相互作用することでエネルギーは熱的平衡状態に至るが、宇宙では粒子密度が希薄なため粒子に加速メカニズムが働いたときに高いエネルギーを粒子が得、非熱的現象の放射を行う。しかしながら加速メカニズムについてははっきりと分かっておらず、さらに詳しい観測が求められている。

非熱的な成分は10 keV以上の領域で観測を行うことが望ましいが、過去に打ち上げられたASCA,Chandra,NewtonなどのX線天文衛星では10 keV以下の軟X線を観測するのが限界であった。10 keV以下では熱的成分が支配的であり、非熱的成分を観測することが難しかったのである。

その状況を打開すべく、2006年に硬X線領域を観測する検出器HXD(Hard X-ray Detector)を搭載したすざく衛星が打ち上げられ、現在活躍中である。そして現在、すざくに続き2013年頃の打ち上げを目指して次期X線天文衛星計画がNeXTが進行中である。NeXT計画には宇宙航空研究開発機構・宇宙科学研究本部(JAXA/ISAS)を中心に、国内外の多数の大学や研究機関が参加している。NeXTには硬X線領域での集光・結像を可能にする多層膜スーパーミラーを用いた硬X線望遠鏡(HXT)が初めて搭載される予定である。HXTの焦点面検出器として硬X線撮像検出器(HXI)が現在開発中である。HXIには30 keVまでの撮像を両面シリコンストリップ検出器(DSSD)が行い、80 keVまでの撮像をCdTe検出器が行う。そのうち、DSSDについては現在広島大学でフライト品の評価をすすめている。DSSDは一枚あたり数10チャンネルから数100チャンネルの信号出力を持つが、HXIの場合では更にそれを複数枚使用する予定なので、そのチャンネル数は数1000チャンネルにも及ぶ。撮像観測を行うには、それらを同時に読み出す必要があり、多数の出力信号を高速に処理をする必要がある。また衛星に搭載することを考えると読み出しを行うシステムには軽量、小型、低消費電力、低ノイズ、高い安定性などといった性

能が要求される。そこで、現在 DSSD の開発と合わせて衛星に搭載することを考慮した読み出しシステムの開発が進められている。

DSSD では多量のデータを高速で処理する必要があり、それに伴ってデータを衛星上で他の装置に高速に転送するための通信インターフェイスが必要となる。衛星では観測機器、処理装置、通信機器などがそれぞれ個別に製作される。従来、これらの機器間での通信規格は衛星ごと、機器ごとに開発されていたため異なったものが混在していた。このことは衛星開発の長期化、コストの増大、信頼性の確保が難しくなるなどといった問題を生じさせていた。このような状況を打開するべく、現在衛星上の各種処理装置等を結ぶ通信規格の統一がすすめられている。現在統一通信規格として SpaceWire が提唱され、世界的にその規格策定が進められている。日本からは JAXA/ISAS を中心に SpaceWire 規格の策定に参加している。SpaceWire は高速通信を行える一方で、可変ビットレートに対応するなど、様々なスケールの装置間での柔軟な接続を実現する。さらに通信インターフェイスを SpaceWire に統一していれば、我々は観測装置の開発だけに集中することができる。NeXT 衛星には SpaceWire の採用が検討されており、そのため NeXT 衛星に搭載される DSSD の読み出しシステムなどのような観測装置は SpaceWire への対応が要求される。

1.2 研究の目的

NeXT 衛星搭載 HXI では DSSD の多量のデータを高速に処理する必要がある。また、DSSD の数 1000 チャンネルにも及ぶ出力信号を deadtime を少なくするためになるべく早く処理を終わらせる読み出しシステムを実現することは難しい。さらに DSSD の原理・基礎特性、読み出しのためのアナログ回路、データ通信を行う装置の制御など広範囲における知識が必要であり、全てを把握しようと思うと長い時間が必要となる。そこで本研究ではそのための基礎研究として SpaceWire を用いた簡単なデータ収集システムを開発した。具体的にはアナログ信号をデジタル信号へ変換する Analog-Digital Converter を搭載した基板を自作し、それをと SpaceWire Digital I/O ボード上にあるプログラマブルな LSI である FPGA を用いてデジタル信号の処理を行った。さらにそのボードの SpaceWire 規格と同じく SpaceWire インターフェイスが搭載された PC である SpaceCube を用いて SpaceWire 規格についての作業を行った。そしてそれらを用いてオシロスコープのように波形を保存する Oscillo System を開発した。

第2章　SpaceWire

科学衛星では観測装置をはじめ、姿勢制御装置、通信装置、各種センサ、データ処理装置など、様々な機器の間で情報をやりとりする必要がある。現在、衛星上の装置間通信のためのハードウェア開発は、それぞれの装置の機能設計と同時に行われることが多い。こうすることでその装置に適した通信装置を開発することができるからである。しかし、それぞれの装置が独自の通信規格を採用するために各機器間通信のためのハードウェアを新たに作る必要がある。そのため、時間がかかり、コストも増大する。さらに、通信のためのハードウェアとそれぞれの装置の機能を実現するためのハードウェアが互いに影響を及ぼし合い、信頼性の低下にもつながる。また、衛星ごとに設計をし直すため、過去の資産も継承されない。このような問題を打開すべく、従来より衛星上の装置間通信規格の共通化が考えられてきた。

衛星上の装置間通信の統一規格として、現在もっとも有力なのが SpaceWire である。SpaceWire はもともと、欧州宇宙機関 (ESA) により IEEE1355 をベースに宇宙用の標準規格として提案された。現在、さらに使いやすいように改良、あいまいさの明確化を行い、組み込み機器間の柔軟な接続を目指している。SpaceWire の規格策定には、ヨーロッパをはじめ世界中の宇宙機関や企業が参加しており、日本からも JAXA/ISAS や大阪大学を中心に参加している。

2.1　SpaceWire の特徴

SpaceWire はデータ転送レートが可変なため、さまざな機器に柔軟に対応できる。1 ラインあたり、2〜400Mbps という高速転送をサポートする。また、伝送には LVDS(Low Voltage Differential Signaling) 方式を採用している。低電圧での差動伝送方式のため、高速通信でも低消費電力、低ノイズでの伝送が可能である。また、DS-LINK を採用し、Data 信号の読み取りに必要な Clock 信号を伝送する Clock ラインを持たないが、Data 信号と一緒に Strobe 信号を伝送することで受信側で Clock 信号を再現することが可能である。図 2.1 にあるように Data と Strobe の XOR をとることで Clock を再現できる。

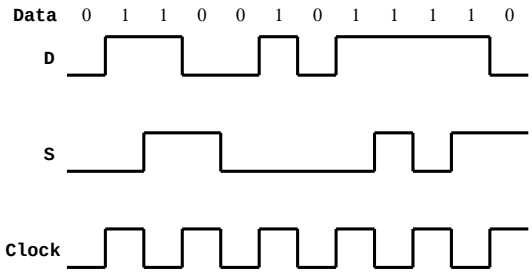


図 2.1: Data-Strobe 信号

SpaceWire は全二重方式を採用しているため、入力と出力を同時に行うことができる。またケーブルは、Data、Strobe が入出力合わせて 2 組で構成されている。それぞれの信号線は強くより合わせたツイストペアを構成しており、ノイズに強い構造をとる。また差動信号ペア間や Data-Strobe 信号間の歪みも小さい。さらに LVDS 方式よりケーブルの長さを最大 10m 以上の長さまで延長することが可能となっている。使用されるケーブルコネクタは 9 ピン D-sub コネクタと同じ形である。このコネクタの型は宇宙空間で使用するのに適任である。

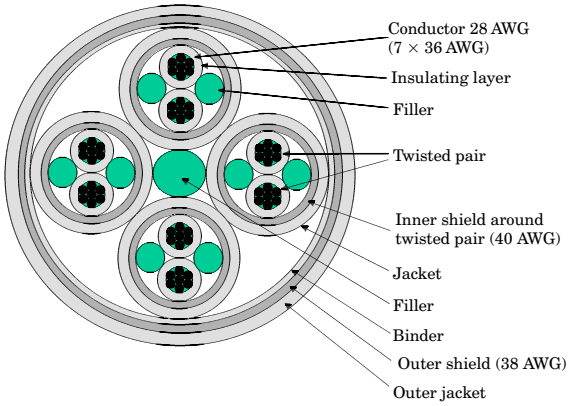


図 2.2: ケーブルの構造 [6]

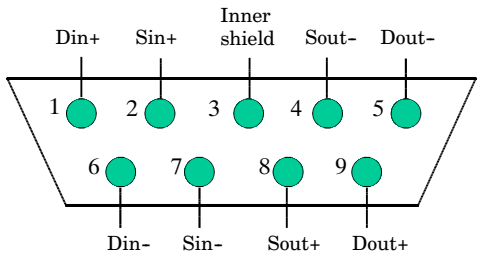


図 2.3: コネクタ・ピンアサイン [6]

2.2 SpaceCube

SpaceCube とは SpaceWire ポートを三つ実装した小型コンピュータのことである。SpaceCube は CPU として MIPS アーキテクチャを持つ NEC 製の VR5701(300MHz) を搭載、PCI バスに LAN、USB を搭載したボードを内蔵する。OS は Linux と T-Engine が対応する。キーボード、マウス、ディスプレイを接続すれば普通の PC として起動することもできる。

小型ながら高機能、低コストでの SpaceWire 動作検証を行うことが可能。以下の表 2.1 に SpaceCube の性能表を、図 2.4 に SpaceCube の概観写真を載せる。

CPU	VR5701 200 MHz / 250 MHz / 300 MHz
Flash ROM	16 MB
DRAM I/F	DDR SDRAM 64 MB
INPUT/ OUTPUT	IEEE1355(SpaceWire),RTC,CF(True IDE),XGA(1024×768),USB1.1, LAN(100BASE),Audio(Stereo), 入出力 RS232C,JTAG I/F(Debug)
POWER	+5 V
SIZE	52 mm × 52 mm × 55 mm

表 2.1: SpaceCube の仕様

図 2.4: SpaceCube の写真

宇宙のような突然の現象などによるシステムの割り込みなどの場合、リアルタイム性に優れた OS の方が対処しやすくなる。これはリアルタイム OS が機器制御を目的としているため、個々の処理内容に優先度がつけられており常に優先度が高い処理から行われるように設計されているからである。人間が相手となる OS ではなく、機器制御という OS ということより T-Kernel は衛星に有利といえる。しかし Linux の場合ではこれまで PC で行ってきた開発環境が有効活用できるため、全面的に T-Kernel が良いというわけではない。すなわち目的、用途に適した OS を選ぶ必要がある。

2.3 Remote Memory Access Protocol(RMAP)

PC 上のプログラムで何かのデータを参照したいというときは、CPU がデータが格納されたメモリからデータを取り出して実行する。このとき CPU はメモリのアドレス空間を参照してデータを取り出す。同じように外部機器との I/O 機能を参照するためにメモリアドレスとは別に I/O アドレスというものがあるが、この I/O アドレスはメモリアドレ

スよりも空間範囲が狭いため、メモリアドレス上に配置して疑似的にI/O アドレスのようにすることができる。このことをメモリマップドI/O と呼ぶ。

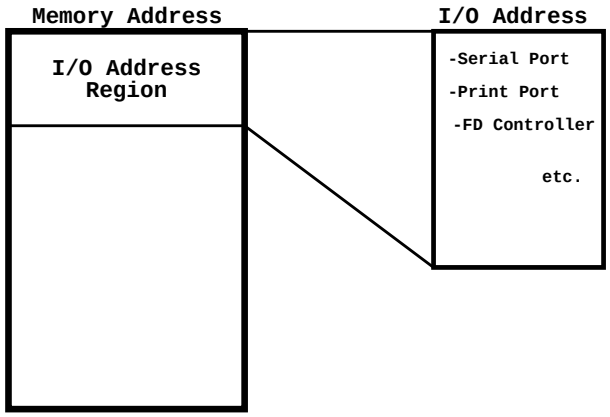


図 2.5: メモリマップドI/O

SpaceWire に接続された機器はこのメモリマップドI/O 機能が使える。そのためSpaceCube 側から見れば自身のメモリにアクセスするかのごとくSpaceWire の先の機器 (本研究ではSpaceWire Digital I/O ボード上のSpaceWireFPGA メモリ) にアクセスすることが可能となる。これによりSpaceWire に繋がった機器にCPU が実装されていなくても制御通信が出来る。このような仕組みを定めたプロトコルのことをRMAP(Remote Memory Access Protocol) と呼ぶ。更に、RMAP を用いればCPU を持たない基板のメモリ情報を簡単にネットワークに組み込むことができる。そのため機器間での複雑なデータ変換のためのハードウェア、ソフトウェアを省略することができ、機器のハードウェアの省スペース、干渉によるデータの損失といった心配が減ることになる。

以上から簡単にネットワーク構成を組み立てることができ、また統一的な規格にすることでデータの信頼性、開発期間の短縮、機器間での通信プロトコルの変換機器を作成しなくてもよいといった様々なメリットがある。つまり開発を全く別々に行っていても通信規格をSpaceWire に統一しておけば簡単に接続でき、プロセッサが設置されていなくてもSpaceWire と接続されていればその機器はリモートメモリアクセスによってその装置のレジスタをさながら自身のメモリを読み取るがのごとく知ることができる。このような特徴から、車載機器や通信機器業界からの関心も高く、民生機器への応用も期待される。

第3章　SpaceWireを用いたデータ収集系の構築

科学衛星では検出器からの多量のデータを処理しなくてはならない。その処理も次なるイベントのために素早く、かつ正確に処理をすることが求められる。更に宇宙という特殊環境に耐えることができ、省スペース、低消費電力であることが重要である。そのため SpaceWire を用いてのデータ収集系の構築は科学衛星の設計においても重要な要素と言える。

この章では主に SpaceWire Digital I/O ボード上のユーザが書き換え可能な LSI である FPGA(Field Programmable Gate Array) を中心に述べる。

3.1　SpaceWire Digital I/O Board

SpaceWire Digital I/O ボードとは今回の実験に用いた SpaceWire 搭載のボードである。アナログ入力が 8 つ、アナログ出力が 8 つあり、FPGA(Xilinx 社 Spartan3 シリーズ XC3S1000) が 2 つ実装されている。SpaceWire と接続されている FPGA は SpaceWireFPGA と呼ぶもので、予め SpaceWire に対応されているように実装されている。もう一方の FPGA は UserFPGA と言うものでユーザによる任意の書込が可能である。これによりアナログ入力ポートからの信号を自由に処理することができる。信号処理を行う Clock は 48 MHz であるが、FPGA 内にある DCM Module(Digital Clock Management Module) にてある程度任意の Clock に変更することは可能である。

現在、大阪大学の能町教授により、FPGA の回路および SpaceCube の Linux 上で動作する SpaceWire 関連のソフトウェアが開発されており、入力されたアナログ信号の波形を SpaceWire のリモートメモリアクセス機能を使って取得できるようになっている。今回、この Digital I/O ボードと SpaceCube を用い、実際に SpaceWire によるデータのやりとりを行い、さらに User FPGA に独自の回路を書き込むことを試みた。図 3.1 に Digital I/O ボードの写真を、図 3.2 に Digital I/O ボードのブロック図を示す。

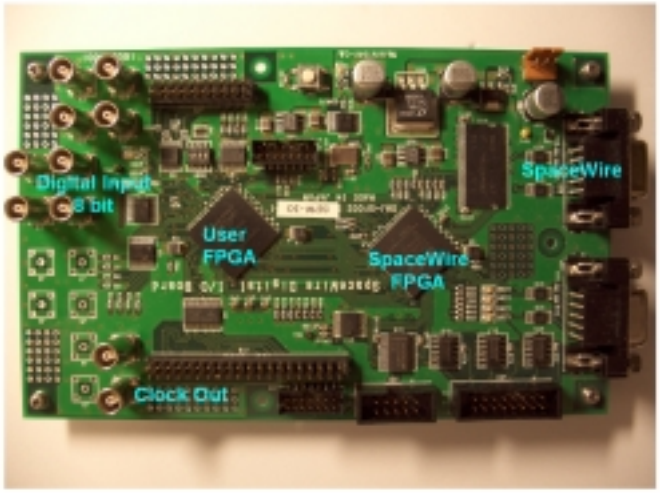


図 3.1: Digital I/O ボード写真

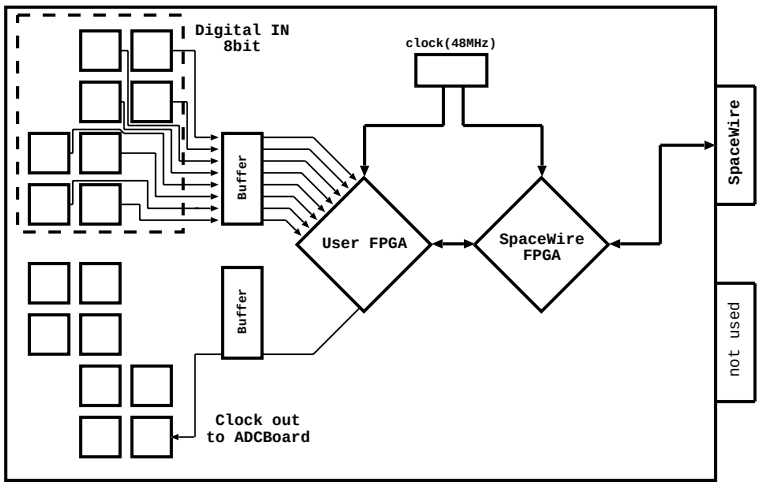


図 3.2: Digital I/O ボードブロック図

3.2 FPGA

システムを開発する際、そのおおまかな仕様設計から始め、より詳細部分へと設計を行う。つまり、抽象度の高いアーキテクチャ・レベルから詳細なゲート・レベルへと段階を踏んで設計を行う。初期のころは真理値表、論理式などを用い、システムの目的に合わせて標準ロジック IC と呼ばれる NOT や AND 回路を含んだ IC を人手により並べていた。しかし、この手法だと実装面積が大きくなる、時間がかかる、他者には分かりづらい、設計回路が正しく動作するかも基板上に組み上げた回路で実際に動作させてみるし

か方法がないなどいった様々な問題があった。そこで、特定用途向けの集積回路であるASIC(Application Specific Integrated Circuit) が使われる。ASIC を作ることで、回路を一つの集積回路に納めてしまい、省面積化、高速化を実現することができる。しかし、このような ASIC は半導体工場で製造されるため、開発に時間がかかり、工場の設備を使用するために莫大なコストもかかる。そのため、ASIC の開発段階での試作や、少量の ASIC しか使用しない場合、その度に半導体工場で LSI を生産するのは非現実的である。また、製造された LSI の中身は書き換えることができないため、後から回路を修正したりすることは不可能である。そこでユーザが自由に回路を書き換えることができるデバイスとして開発されたのが FPGA である。

FPGA は ASIC のように工場に注文して作成する形式ではなく、回路を設計する設計者の手元で作成した回路を書き込むことができる LSI である。書き込むことをダウンロードと呼ぶが、ダウンロードに必要な機材は FPGA の他にダウンロードケーブルと PC ぐらいなので簡単に実装することができる。似たようなもので回路を書き込むことが出来るデバイスで CPLD(Complex Programmable Logic Device) というものがあるが、一般に FPGA の方が大規模回路に適している。

3.2.1 Spartan-3

ここでは Xilinx 社の Spartan-3 シリーズ FPGA を用いて FPGA の内部構造について述べる。他のシリーズでも基本的な構造は似ている。

FPGA の内部には CLB(Configurable Logic Block) という回路を実装する上で最も重要なロジックリソースが規則正しく敷き詰められている。各 CLB には 4 つのスライスがあり、各スライスには 2 つの LUT(Look Up Table) が含まれる。図 3.3 に CLB の配置について示す。

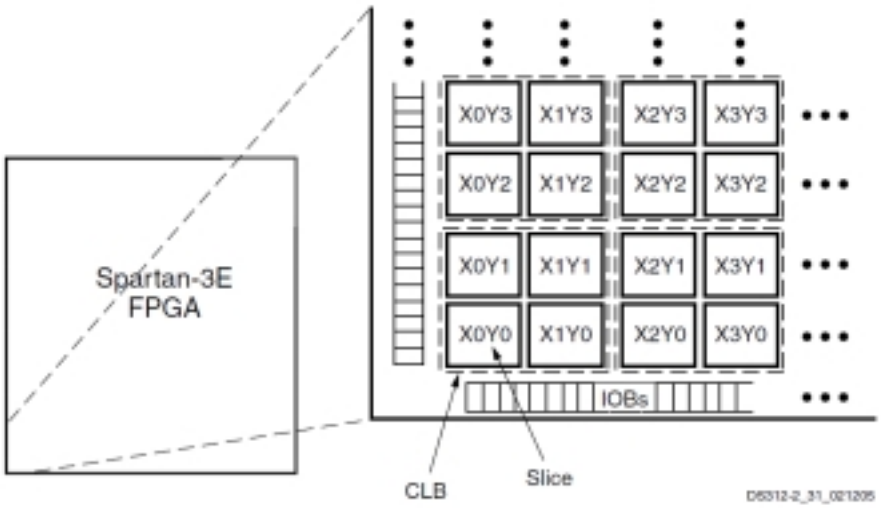


図 3.3: CLB Location [13]

CLB 内の 4 つのスライスは二つずつグループ化されており、二つのグループが列になるように組まれている。このうち左のペアを SLICEM と呼び、右のペアを SLICEL と呼ぶ。SLICEM ではロジックファンクションとメモリファンクションがサポートされているが SLICEL ではロジックファンクションのみとなっている。しかしパフォーマンスの面では SLICEL の方が SLICEM より優れている。図 3.4 に SLICEM を簡略化した図を示す。

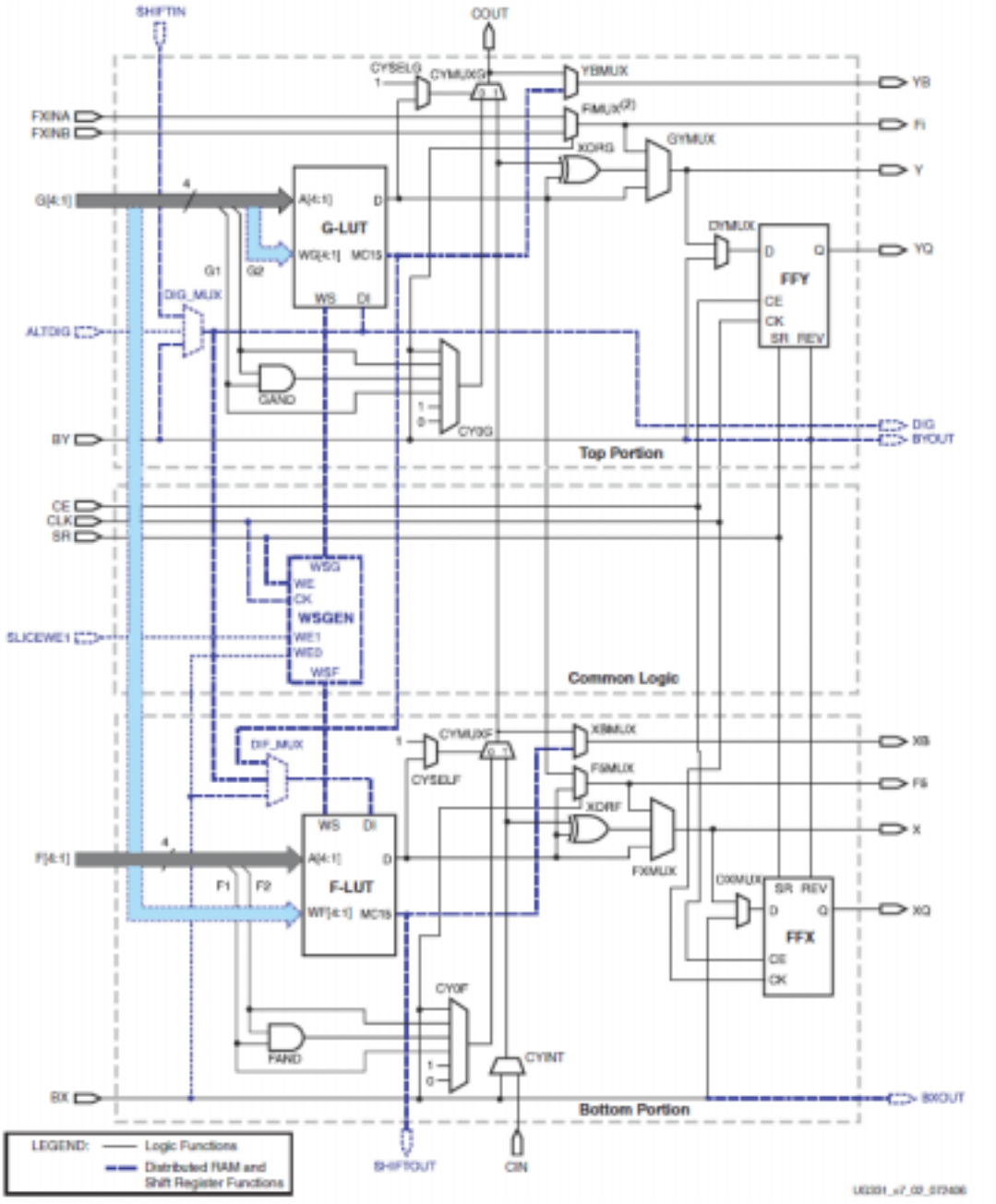


図 3.4: 左側の SLICEM を簡略化した図 [13]

この LUT と右に描かれた FF(Flip Flop) の組合せをロジックセルと呼び、この組合せを変えることでユーザの希望する回路を実現する。つまり CLB は回路ロジックの性能を支えるということである。

また、各 CLB を結ぶ信号経路をインターコネクトと呼び、この経路を変更させることで自由な回路設計を実現する。インターコネクトは自由に配線を行うために分割され、様々

な種類に分かれている。ロングライン、HEX ライン、ダブルライン、ダイレクトラインの四種類である。これら四つのラインは接続できる距離、接続する際の駆動条件、本数、特性が異なり、これらを組み合わせることで様々な論理回路を実現する。

一般的な FPGA の場合、回路情報を書き込むときは FPGA 内にある SRAM(Static Random Access Memory) に書き込む。SRAM は DRAM(Dynamic Random Access Memory) と違い、リフレッシュが不要で、記憶保持の電力が少なくてすむ。しかし SRAM は揮発メモリなため、電力の供給が途切れると記憶内容が失われてしまう。そこで電力が途切れても記憶内容が失われない ROM に書き込みを行い、電源投入時に自動で読み込みを行えるような ROM を準備しておく。この ROM をコンフィギュレーション ROM と呼び、SpaceWire Digital I/O ボードの場合、FPGA の外部に設置している。Spartan-3 シリーズのコンフィギュレーション ROM は 20000 回までのプログラム、消去が可能である。

3.2.2 UserFPGA と SpaceWireFPGA

Digital I/O ボード上でユーザが実際に書き込みを行うことができる FPGA が UserFPGA である。信号処理等の Module を書き込むのがこの FPGA である。UserFPGA では予めその雛型と言うべき形が作られているため、新たにユーザが作成するのはそのうちの実際の処理を行う Module ぐらいである。

ユーザが関与できない FPGA が SpaceWireFPGA である。SpaceWireFPGA 上には図 3.5 のようなアドレス空間が構成されている。

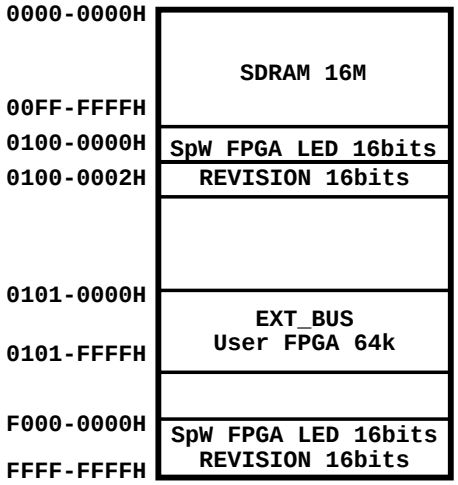


図 3.5: SpaceWire Digital I/O ボード上のメモリ空間 [3]

ユーザが SpaceCube から UserFPGA で処理した信号を読み取りたいといった状況の時は、SpaceCube 側から SpaceWireFPGA アドレス空間 0101-0000H 番から 0101-FFFF 番

にアクセスして読み取りを行う。つまり、SpaceCube 側からは直接 UserFPGA を見ることができない。0101-0000H 番から 0101-FFFFH 番までのアドレス空間は UserFPGA と繋がるためのアドレスであり、ユーザはここに値を書き込んだり、読み取ったりといったことしかできないということである。ただし UserFPGA で、どのメモリアドレスが UserFPGA のどのレジスタと対応しているか設定する必要がある。その設定ファイルを AddressMap とする。AddressMap では 0101 以下のアドレスと UserFPGA での信号などとの対応を指定する。例えば SpaceCube 側から 0101-20acH 番目のアドレスを Read しろといった命令が入ったとする。SpaceWireFPGA はそのアドレスが 0101-0000H から 0101-FFFFH の間なので UserFPGA への命令であると判断して命令を UserFPGA へ流し、UserFPGA は AddressMap から 0101-20acH は信号 A であると判断する。そしてその信号 A の値を SpaceWireFPGA へと返し、SpaceCube がそれを読み取るというわけである。図 3.6 に SpaceCube からの信号から UserFPGA がどの信号のことかを特定するまでの過程を描いている。

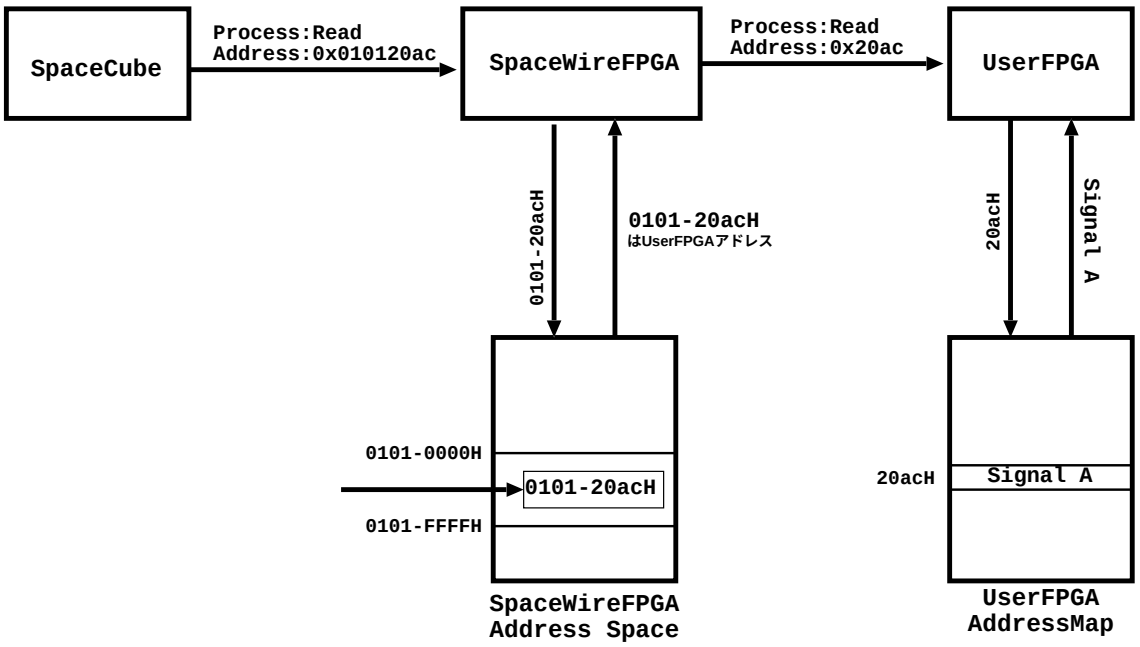


図 3.6: SpaceCube からのアドレスがどの信号なのか User FPGA が特定するまで

3.2.3 VHDL

少し前まではハードウェアを目的の動作に向けて記述する際、真理値表、論理式などを用いて AND や OR 回路を組み合わせたゲートレベルでの設計が主流だった。つまり、ある処理を作動させようと思ったらどのように処理をするかを'0' と'1' を用い、更に何かし

らの条件を考えるときでさえ AND や OR などの論理式を考えなくてはならなかった。これでは高度な処理を記述しようとすればするほど回路が複雑になり、設計にかなりの時間がかかる。誤作動が起きたときにエラーの判別がしにくくなり、更に設計者以外にはその回路の意図を掴むのが難しくなる。

これを解消するために、著しく発達したコンピュータを利用して回路の接続関係を記述する言語、ハードウェア記述言語 (HDL:Hardware Description Language) が利用されるようになった。HDL ではゲートレベルでの記述以上に抽象度の高い記述を行うことができる。つまり C 言語のようなコンピュータ言語を操るかのように回路を記述することが出来る。これにより回路を動作に合わせて設計しやすくなり、他人にも動作内容が分かりやすくなる。誤作動が起きたときも対処しやすく、回路変更も容易となる。

現在最も普及している HDL には VHDL と Verilog HDL がある。本研究では文法的には難しいが高い記述能力を持つ VHDL(VHSIC HDL) を用いて回路設計を行った。VHDL は、米国国防総省の VHSIC(Very High Speed Integrated Circuit) 委員会によって提唱されて開発された。当時、国防総省向けの ASIC を開発する際ゲートレベルでの開発を行っていたため開発に長い時間がかかり、設計を開始したころの最新半導体部品を基準に開発していたとしても設計が完了するころには既にそれ以上の部品が出来ているため完成品が時代遅れとなってしまうという問題があった。そのため、より抽象度の高く、開発終了時にその時点での最新の部品が使えるような設計手法を開発する必要があった。これより、1981 年に提唱、1986 年に公開されたのが VHDL である。その後文法の改訂作業を経て、IEEE(米国電気電子技術者協会) により標準化され、VHDL は世界の標準 HDL として認識されている。現在、LSI のような大規模設計を行うことができるようになったのも VHDL などのような高度のハードウェア記述言語のおかげとも言える。

VHDL は様々なレベルでの記述が可能である。抽象度の高いシステムの概要を記述したり、下のレベルである装置間信号の記述、モデル化を行ったモジュールの記述といったことも可能である。更に下のゲートレベルでも記述を行うことが出来る。VHDL でロジック回路の生成を行うためには RTL(Register Transfer Level) で記述する。このレベルで記述された場合、ロジック回路を簡単に論理合成して作成することが可能である。以下に VHDL によるロジック・ゲートの記述、図 3.7 にその論理合成の結果を載せる。

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sample is
        port (     A : in std_logic;
                   B : in std_logic;
                Out1 : out std_logic;
                Out2 : out std_logic);
end sample;

architecture RTL of sample is

signal sig : std_logic;

begin

         sig <= A nor B;
        Out1 <= not sig;
        Out2 <= A and sig;

end RTL;
```

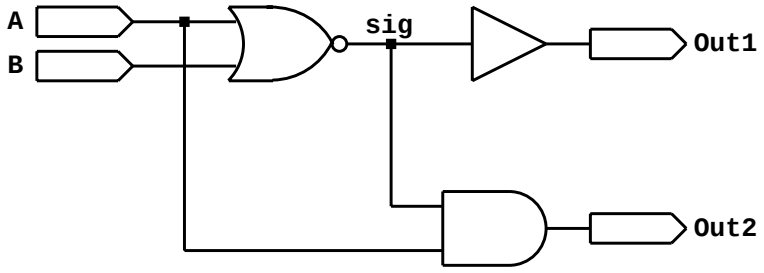


図 3.7: VHDL によるロジック・ゲート例

VHDL は高い記述力を誇り、大規模回路、かつテキストを変えるだけで簡単に回路を変更することが出来る。この VHDL を自在に使える開発環境として Xilinx 社が無償で配布している総合開発ソフト、ISE 9.2i を用いた。このソフトは DVD によってインストールを行ったが、ISE WebPACK としてインターネットからダウンロードすることもできる (http://japan.xilinx.com/ise/logic_design_porod/webpack.htm)。ISE は VHDL のデザイン、論理シミュレーション、論理合成を行うことができる。更に回路を書き込むデバイスを適切に選択することで、配置配線、タイミングシミュレーション、更に実際のデバイスへのダウンロードも可能である。すなわちこれ一つでデザイン設計から実装まで全て行うことができる。

本研究ではより高度なシミュレーションを行うために、同じく Xilinx 社の配布である ModelSim Xilinx Edition-III/Starter 6.2g(以下 ModelSim XE-III Starter) を用いた。この Starter 版は無償ながら ISE 付属のシミュレータより高度な機能を実装している。使

用ケースはCPLDデバイス、小規模FPGAなどが対象なため、より高度な機能をシミュレーションするのならば有償のXilinx社提供ModelSim Xilinx Edition-IIIか、同じく有償のModel Technology社提供ModelSim PEなどを用いればよい。本研究ではModelSim XE-III Starterで十分であるためこちらを用いる。ModelSim XE-III StarterはISEの環境に対応し、ISE側から簡単にシミュレータをModelSim XE-III Starterに変更することが可能である。ModelSim XE-III Starterはタイミングチャートのようなシミュレーションを行うことが目的だが、その内部での信号の様子を知ることが出来る。更にISEのシミュレーションで使用可能なロジックゲート数限界以上を扱うことができるため、より規模の大きい回路でもシミュレーションを行える。このISE 9.2iとModelSim XE-III Starterを用いて本研究の開発を行った。

3.3 DSSDの読み出しへの応用

本研究では基礎実験が目的だが、将来的にはFPGAとSpaceWireを用いてDSSD(Doble-sided Silicon Strip Detector:両面シリコンストリップ検出器)と呼ばれる検出器の多チャンネル読出しを目標とする。ここではFPGAの応用として簡単にDSSDとはどういうものか述べる。

3.3.1 DSSDの概要

DSSDは次期X線天文衛星であるNeXT衛星に搭載予定の半導体検出器である。NeXTに搭載される四つの検出器の内、硬X線撮像検出器(HXI)部にCdTeピクセル検出器と共に搭載され、DSSDは5-30 keVの領域を検出することが出来る。

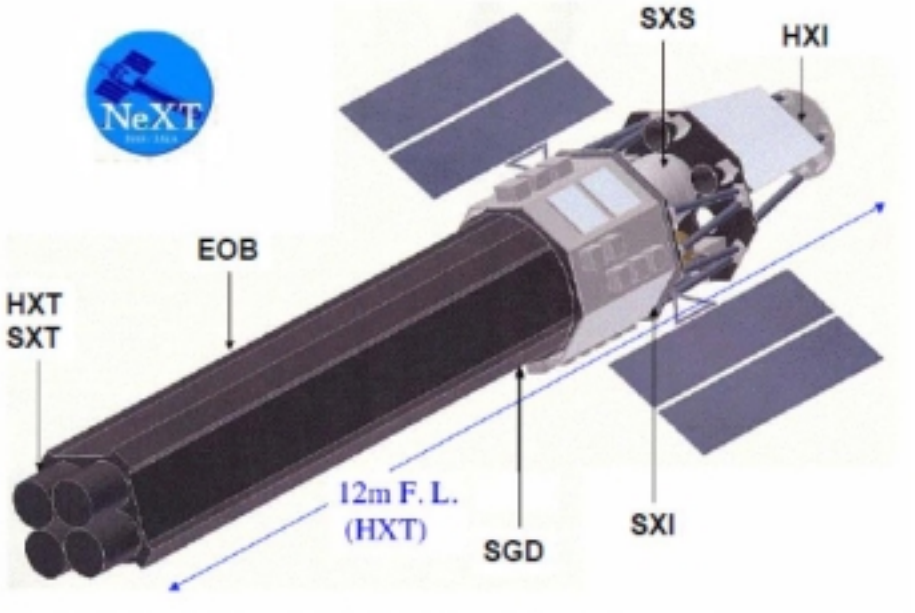


図 3.8: NeXT 衛星と搭載される検出器 [7]

DSSD というのは n 型シリコンウェハー片面に p+ストリップを、もう片面に n+ストリップを直行するように形成した検出器である。両ストリップの交差した領域は独立のダイオードとして働くため、擬似的に多量のダイオードが碁盤の目上に配置したかのように見える。このダイオード構造に逆バイアスをかけ、キャリアをのぞいた空乏層を生成する。この空乏層に放射線が入射すればその放射線からエネルギーを得た束縛電子は様々な過程を経て束縛を振り切り、電子と正孔の対を作る。この電子・正孔対ができると付加した電圧によって電子は n+側へ、正孔は p+側へと移動する。つまり一つの放射線でも同時に二つの検出を行うことが可能ということである。n+ストリップと p+ストリップは直行しているため、検出した情報から放射線が入射した位置の二次元情報を得ることができる。

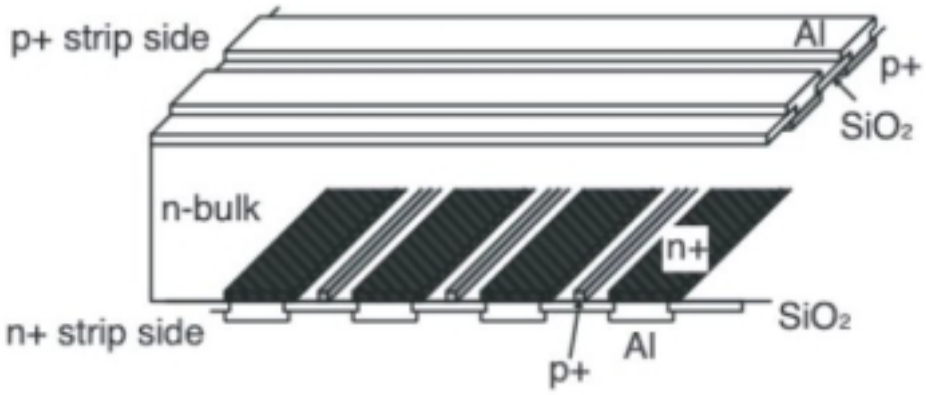


図 3.9: 両面シリコンストリップ検出器 [8]

3.3.2 DSSD と FPGA

DSSD に放射線が入射したら、その放射線強度は入射したダイオードの空乏層で発生した電子・正孔対の多さ、最終的には電圧の大きさで分かり、その位置は検知したストリップから座標のように知ることができる。しかしそのダイオード部分だけ調べるのであれば座標を知る必要はない。つまりわざわざ DSSD にする必要はない。DSSD の良さというのは多数のストリップを利用し、放射線の二次元位置情報を得ることである。そしてその情報を知ることができるダイオードが多数あるのであればそれらを放射線が来たと同時に全て読出しを行うことで放射線全体がこの DSSD へとどのように入射したのかといったことが撮像できる。ストリップ間隔は 250 um で、この間隔がピクセルサイズになるため、かなり詳細な画像を撮ることが可能である。

しかし全て読出しを行うといってもそのチャンネル数は膨大である。NeXT 衛星の HXI 部にには DSSD は一枚のみ搭載されるのではなく、図 3.10 のように何枚も重ねてセットされる。

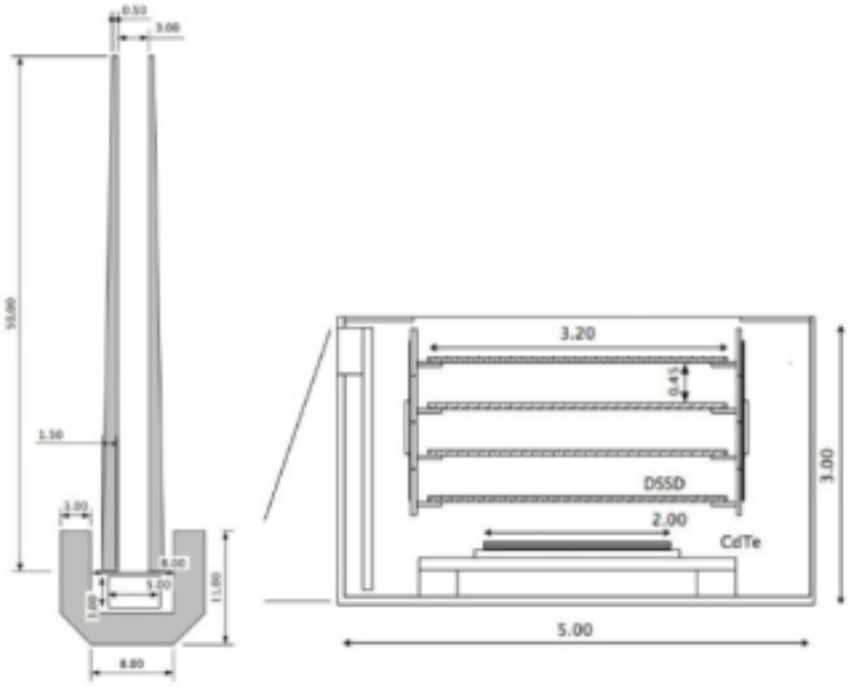


図 3.10: HXI の概念図 [7]

更に一枚の DSSD はストリップ数が n+側、p+側それぞれ 128 枚あるため、一枚につき 256 枚のストリップがある。更に実際にはこのような DSSD が数枚ある。例えば図 3.10 のように 4 枚あれば 1024 チャンネルを同時に読み出す必要がある。これだけ多数のチャンネルのデータを読み出すため、単純にデータを取り出すだけではなく、より柔軟なシステムによって読み出すことができるのが理想である。そのために用いるのが FPGA である。プログラマブルなロジックデバイスにより、FPGA 内でのモジュールがある程度独立に働くため、柔軟な読み出しを行うことができる。更に VHDL によって簡単に複雑なハードウェアを構成することができるため多量のチャンネル数の処理を行うのも簡単である。

DSSD のような様々な状況に応じてモジュール変更を行わなければならない場合などでも FPGA ならば可能となる。プログラマブルなため、仮に違うシステムへと使わなければならないときでも簡単に内部回路を変更することができるため経済的でもある。

第4章　SpaceWireによるADCボードの読み出し

検出器からの信号はアナログ信号であり、コンピュータで解析を行うにはデジタル信号でないといけない。そのためにアナログ信号をデジタル信号に変換する必要がある。更にデジタル変換しただけではどのような信号が分からないため、SpaceWire Digital I/Oボードに入力したデジタル信号を目的に合わせて処理する必要がある。そのために本研究ではADC(Analog-Digital Converter) ボードを製作し、それをSpaceWire Digital I/Oボードと接続して信号を処理、その波形を保存するOscillo Moduleを開発した。このボードによりアナログ信号をデジタル信号に変換し、FPGAで信号処理を行う。ADCボードからLANに接続するまでの過程を簡単に次の図4.1に載せる。

図 4.1: ADCボードからLANまでの簡単なセットアップ図

LANに繋いでいればSpaceCubeに接続していないPCでも解析を行うことが出来る。本研究ではシリアルコンソールホストPCと同じLinuxPCでLAN経由で得たデータを処理したが、同じLANに接続していれば全く別のPCでも処理を行うことが可能である。

4.1　Analog-Digital Converter

ここではアナログ信号をデジタル信号へと変換する過程を示す。アナログ信号は連続的な電圧であり、デジタル信号は離散的な値である。この変換には専用のチップを用いる。そのアナログ信号をデジタル信号に変換するためのチップとしてBurr-Brown社のADS831を用いた。このチップはサンプリングレートは最大80MHz、ビット分解能は8bitの能力を持つ。つまり12.5ns間隔でのサンプリングが可能であるため、放射線が入射したとき

の検出器からの信号にも対応することができる。さらに単電源で動作するため、省スペースである。

4.1.1 パイプライン型 ADC 変換原理

ADS831 の変換原理はパイプライン型と呼ばれるもので、高速変換、高分解能の両方を実現している。そのため、検出器からの信号のような微弱な信号で、かつ高精度のサンプリングが要求された場合でもそれに応えることができる。

変換方法は上位 bit から順に変換していくという方法である。まず大雑把に大体の大きさを測り、その差の部分をより詳しく調べるという、シンプルな方法と言える。図 4.2 にイメージ図を載せる。

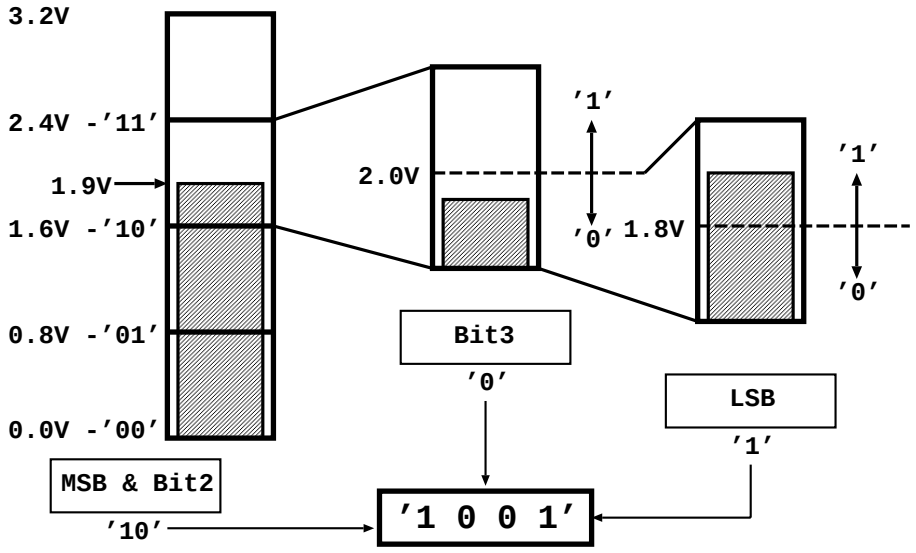


図 4.2: パイプライン型 ADC の基本変換原理

この図では入力電圧 1.9V で調べている。測りの基準といえる上位 bit の比較用電圧は今は 0.0 V、0.8 V、1.6 V、2.4 V とする。まず上位ビットを調べる。1.9 V は 1.6 V より大きく、2.4 V より小さいので上位二つの bit 数は'10' となる。次にこの 1.6 V から 2.4 V までの間を詳しく調べる。この二つの中心電圧は 2.0 V。この電圧より 1.9 V が大きいか比較するとやはり小さいので Bit 3 は'0'。更に細かく調べて LSB を求める。この 1.6 V と 2.0 V の中心電圧は 1.8 V。この電圧より入力電圧は大きいので LSB は'1' となり、全てまとめるとこの場合 1.9 V は'1001' となる。パイプライン型はこの方法をもう少し回路の組みやすい方法に変化した変換方法であるが基本原理は同じである。

パイプライン型というのは、サンプルホールドアンプ (S/H)、低分解能なフラッシュ型 ADC、DAC(Digital-Analog Converter)、そしてアナログ減算回路 (SUB) がセットになっ

たブロックを複数直列に配置した回路の形である。ビット数の数が増えればその分このブロックの数も増加する。図 4.3 パイプライン型 ADC のブロック図をを示す。

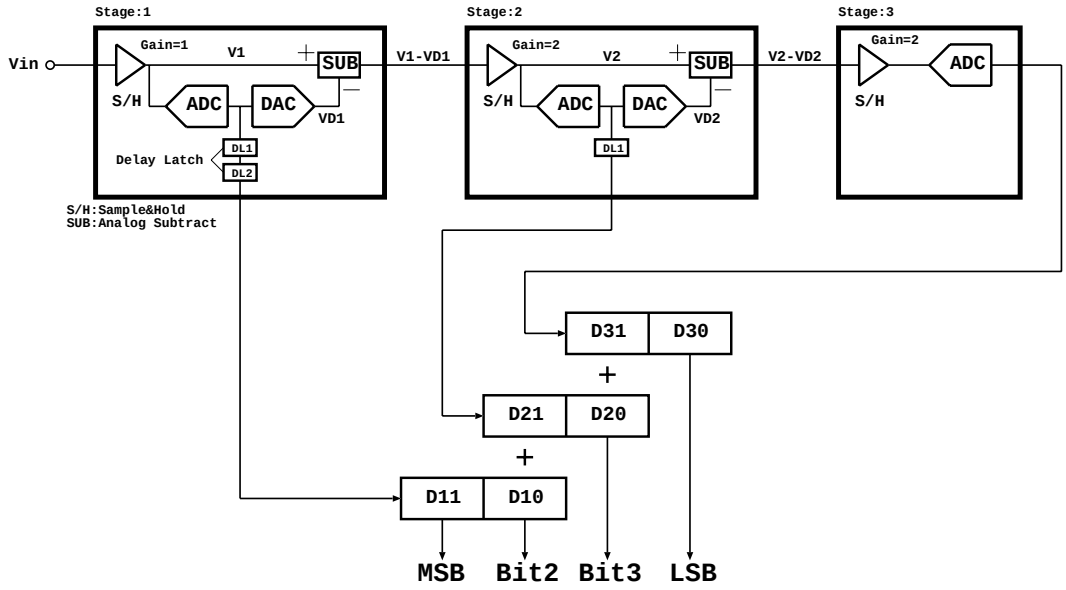


図 4.3: 分解能 4bit のパイプライン型 ADC ブロック図

一つ目のブロックでは入力された信号を S/H で固定し、その電圧をフラッシュ型 ADC で決定する。フラッシュ型はパイプライン型よりも低分解能だが高速で変換が可能である。変換方法はパイプライン型より至って単純であらかじめ段分けされた電圧レベルより高いか低いかで決定する。この場合仮にフラッシュ型 ADC が 2bit の分解能であるとするとデジタルで 2bit が出力される。しかしまだ下位 bit が決まっていないので全ての bit が決定するまで待機する必要がある。そのためにディレイラッチを必要数置き、デジタル信号を留めておく。2 bit の ADC では元の入力電圧との差があるのでそれをより詳しく決定するために SUB で差を出し、次のステージへと渡す。

次のステージではより詳細に決める。図 4.2 では 1.6 V と 2.4 V の中心電圧を求めてその電圧と入力電圧を比べる方法だったが、ここではその余りの部分を S/H で倍にすることで代用する。そして 0.8 V 以上なら'01' であることを利用し、そこで 3 bit 目が'1' か'0' なのかを決定する。こうすることで同じ部品を使うことが出来、かつ S/H で差を倍にさえ出来れば次のステージでも同じ比較をとることができるので。フラッシュ型には出来ない高分解能の ADC を作ることができる。図 4.4 に図 4.2 を実際のパイプライン型 ADC で変換するとどうなるか示す。

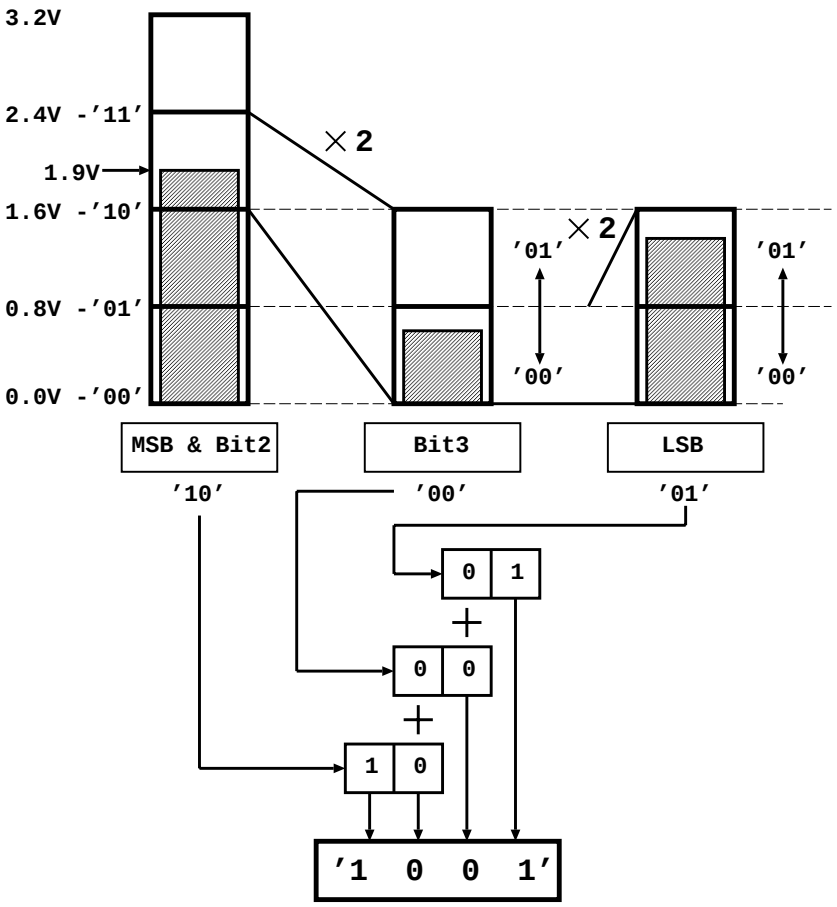


図 4.4: パイプライン型 ADC の変換手順とその結果の出し方

この方式は入力されたクロックに合わせてデータをコンバートする方式である。ただしさきほど述べたようにこのパイプライン型は直列に配置した複数のブロックから成る。各ブロックは 1 クロックで一つ bit を吐き出す (上位 bit のみ二つ) という方法であり、また、パイプライン型の特性からデータを確定させるためには全てのステージを終了しないといけない。つまり入ってきた信号は 0 クロックで出力されるわけではなく、出力データは入力信号より遅れて出力される。これをデータレイテンシと呼び、ADS831 の場合だと、4 クロック分のずれ、つまり 40MHz でサンプリングすれば 100 ns のずれが発生する。精密なシステムを構築する場合、このずれを考慮しなくてはならないときは無視できない。パイプライン型 ADC のタイムチャートを簡単に描くと図 4.5 のようになる。

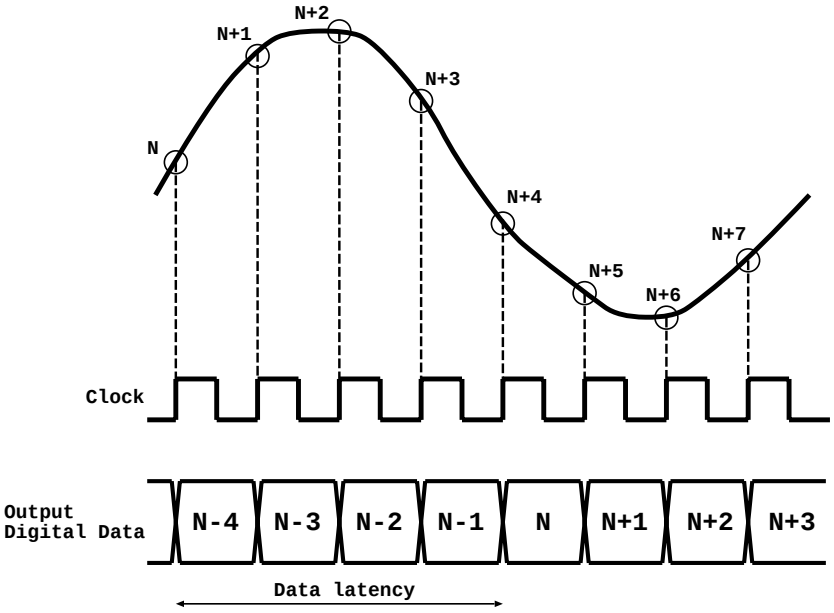


図 4.5: パイプライン型 ADC タイムチャート

本研究では最終的には波形保存が目的だが、波形が発生したナノ秒単位での正確な時刻を調べる必要がないため、データレイテンシについては深く触れない。

この ADS831 を使うために入力電圧の調整、出力・クロックの緩衝等を行う回路を設計する。

4.2 ADC ボードの開発

ADS831 を用いて、入力されたアナログ信号をデジタル信号へと変換するための ADC ボードを製作した。このボードで FPGA に入力するための ADC を全て行う。ADC ボードは次のような仕様で作成することを目標にした。

- ビット分解能を 8 bit にする
- サンプリングレートを 48 MHz とする
- 出力信号は Digital I/O ボード Input 部に合わせて 3.3 V で出力する

4.2.1 ADC ボード部品

ADC ボードの電源供給は松定プレシジョン株式会社の直流電源 PLD-36-1.2 を用いた。この直流電源は電圧が-36 V から+36 V まで、電流が-1.2 A から+1.2 A まで供給できる。

今回の実験では電圧は最大でも+5.0 V あれば十分なのでこの直流電源は適すると言える。しかし回路では直流電源が供給できる+5.0 V、-5.0 V 以外に+3.3V の電圧が必要となる。5.0 V 用の電源とは別に、もう一つ 3.3 V 用直流電源を準備してもよいが、それでは回路周辺がケーブルで繁雑になる。また直流電源をそのためだけに増やすと無駄な実験スペースが広くなる。そこで+3.3 V は+5.0 V の電圧レベルを調整することで供給できるようにした。そのために用いたのが三端子レギュレータである。実験には TOSHIBA 社の TA48M033F を用いた。

SpaceWire の入力ポートが 3.3 V に対応しているため、ADC ボードの出力信号を 3.3 V にし、入力信号が ADS831 の ADC が可能な電圧領域にするために OP Amp を用いて Offset 電圧をかけた。この ADS831 の変換可能電圧をリファレンス電圧と呼ぶが、ADS831 からリファレンス電圧のトップ電圧とボトム電圧が出力されているので Offset 電圧はこのボトム電圧を利用した。クロックは余裕をみて 24 MHz のクロックを入力した。図 4.6 に ADC ボードのブロック図を示す。

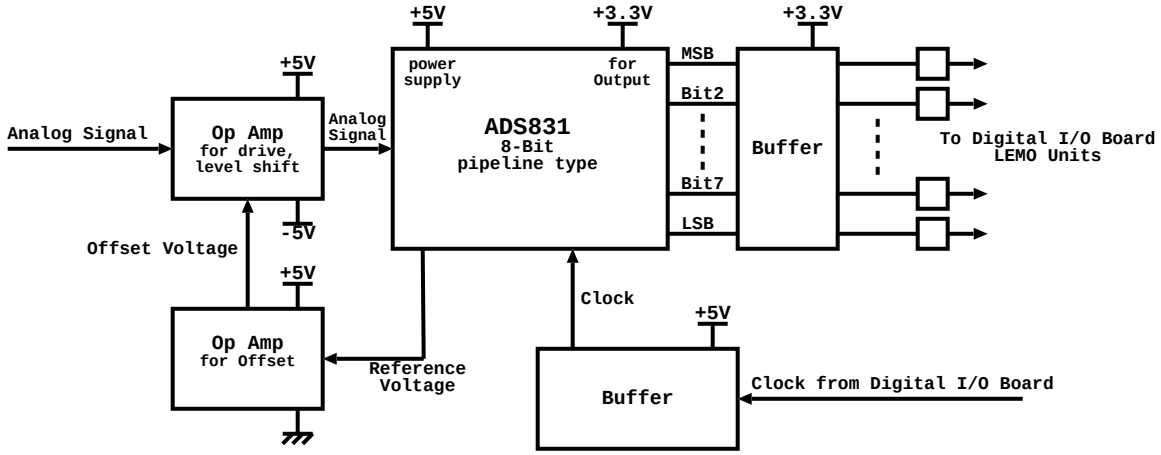


図 4.6: ADC ボードブロック図

ここで ADS831 の出力信号や、FPGA からのクロック信号のドライブ能力の小ささを補うための緩衝目的でバッファを一つずつ設置した。バッファは TOSHIBA 社 TC74LCX541F である。このバッファは ADS831 の出力信号電圧の 3.3 V、FPGA からのクロック信号電圧の 5.0 V の両方に対応しているので両者とも同じものを用いた。

OP Amp は Burr-Brown 社 OPA691 を二つ用いた。アナログ入力信号の緩衝・増幅目的のための OP Amp は正電源に+5.0 V、負電源に-5.0 V かけた。こうすることで OP Amp のアウトプット電圧範囲が-5.0 V から+5.0 V までとなり、単電源での動作のときと比べて格段にアウトプットの電圧範囲が広がる。Offset 用の OP Amp は入力信号の OP Amp のときほど出力電圧範囲が幅広いものである必要はない。これは Offset 電圧は入力信号の電圧レベルの基準を底上げするのが目的であり、電圧変化に対応するのが必須の性能で

はないからである。OPA691 は単電源、すなわち OPA691 の正電源に+5.0 V をかけ、負電源を GND とした場合、出力電圧範囲は 1 V から 4 V である。Offset 電圧を入力信号の電圧範囲によって変化できるように設定できるのが望ましいので、Offset 用 OP Amp の電圧調整としてポテンシオメータ (1 kohm) を一つ設置した。こうすることで Offset 用の電圧を 0 V から 1.5 V まで変化できるようにした。

電源となる電圧は変化せずに安定に供給されることが望ましい。そこでその役目を担うためにバイパスコンデンサを電源入力部すぐそばに並列となるように設置した。バイパスコンデンサというのは、電源電圧からの電気を溜め込み、電源電圧からの電圧が不安定になった場合、それを安定させるように働くコンデンサのことである。今回開発した ADC ボードには、OP Amp の+5.0 V 部、-5.0 V 部、ADS831 の+5.0 V 部、+3.3 V 部、バッファの+3.3 V 部、+5.0 V 部の全てに 0.1 uF のコンデンサを回路パターンに並列となるように、また各端子になるべく近くなるように設置した。

電圧グラウンドは ADC ボード入力前のアナログ部と入力後のデジタル部で大まかに分けた。ただし両者も完全に独立というわけではなく、一点でつなげている。この一点もどこでもよいというわけではなく、なるべく直流電源のグラウンド線に近くなるように決定した。これは一点アースと呼ばれる方法で、こうすることでグラウンド間での電位差が小さくなる。すなわち電圧降下による電流の流れ込みを防ぐことができる。デジタル信号は High の状態か Low の状態の二通りでの出力のみで安定させることが必要なため、グラウンドの不安定からそれ以外の状態になることは避けなければならない。

図 4.6 から ADC 全体の回路図を設計した図が図 4.7 と図 4.8 である。

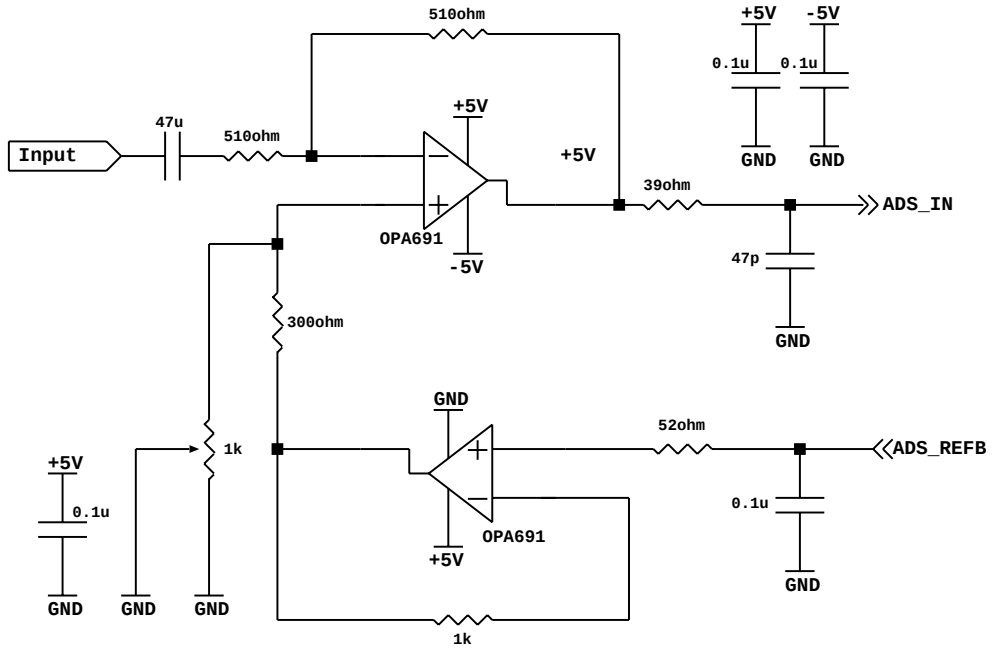


図 4.7: ADC ボードアナログ信号部回路図

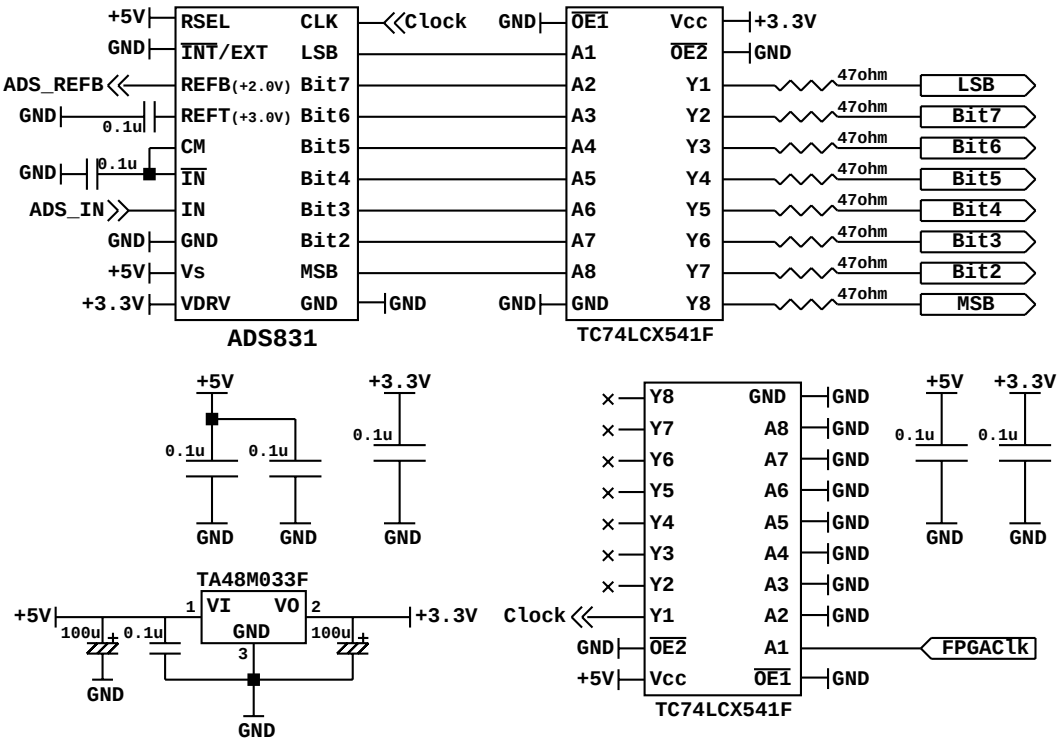


図 4.8: ADC ボードデジタル信号部回路図

図 4.7 はアナログ信号の処理部である。この回路でアナログ信号を ADS831 の ADC 可能範囲内に電圧が入力できるように調整する。図 4.8 では実際に ADS831 が ADC を行う回路を示している。直流電源から得ている 5 V 以外の電圧がこの回路では必要なので TA48MO33F を用いて 3.3 V を供給している。ADS831 の RSEL というのはアナログ信号の入力電圧範囲を選択できる端子である。ここが High なら ADS831 は IN 端子に入った信号のうち、電圧が+1.5 V から+3.5 V まで ADC を行う。Low ならば入力電圧の+2.0 V から+3.0 V まで ADC となる。今回はよりなるべく汎用的になるように作るので RSEL に+5 V をかけ、High の状態にした。Vs というのはこの ADS831 を稼働させるための供給電源であり、VDRV というのは出力信号の電圧レベルを決めるものである。Digital I/O ボードは+3.3 V で入力するのが理想であるため、ADS831 のあとに緩衝用として+3.3 V に整合するバッファがあるが、ここでも+3.3 V で出力するように VDRV に+3.3 V をかけた。

これら二つの回路を一つのボードに集約して実装した。この図 4.7、図 4.8 を元に実際に基板へ実装するデザインを決め、ADC ボードを作成する。

4.3 ADC ボードの製作

まず始めに回路図をもとにプリント基板への配置を考えなくてはならない。用いたプリント基板はサンハヤト社のポジ感光基板 10K を用いた。これは片面用で大きさが縦 75 mm、横 100 mm の大きさのため、ADC ボードに用いる部品の数からしても大きさはちょうどよい。

4.3.1 PCBE

基板への回路の設計には、高戸谷 隆氏がインターネットネット上でフリーソフトウェアとして公開している PCBE(http://www.vector.co.jp/soft/win95/business/se056371.html) を用いて描いた。PCBE はプリント基板、版下作成用作画ソフトで、ガーバファイルを作成することもできるため、プリント基板メーカに発注することもできる。さらにパターンの太さも 0.1 mm から選べ、二層以上のプリント基板を作成することもできる。レイヤの種類も豊富で、印刷するときにどのレイヤを印刷するか選ぶことができるため、部品など、プリント基板に印刷しないが回路を製図するときにはあったほうが作りやすい、というようなものも表示しながら作図できる。またベクタイメージなので拡大しても作業をしやすい。図 4.9 に PCBE で作成した ADC ボードのイメージ図を示す。

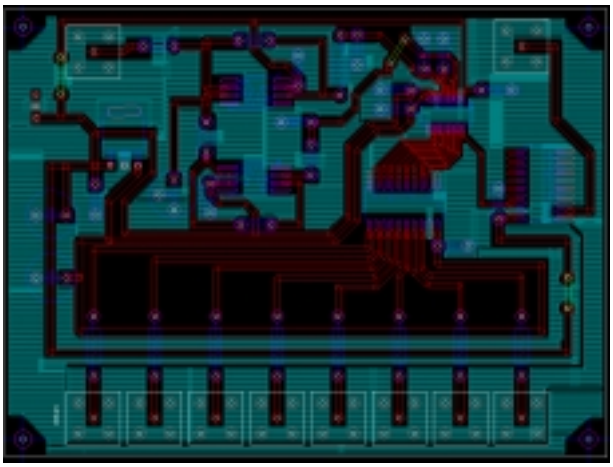


図 4.9: PCBE

PCBE での描画が完成した後にプリント基板へと感光印刷させるための版下印刷をする。感光基板は、基材上に導通部である銅箔面、その上に感光被膜という三層構造をしている。この基板に露光、現像、エッチング作業を経ることで目的の回路を作成することができる。図 4.10 に感光基板の図を示す。

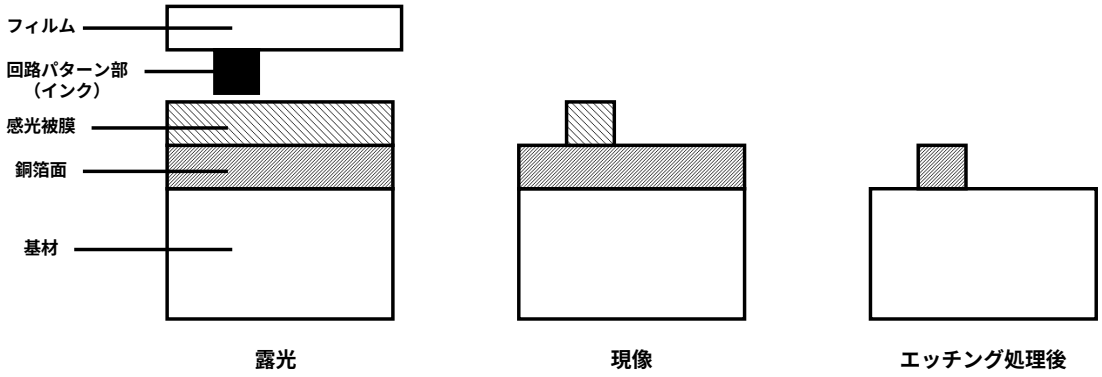


図 4.10: 感光基板

光を基板に露光させると、光が当たった感光被膜が化学反応をおこす。詳しい過程は後に説明するが、最終的に化学反応をしなかった箇所は導通部に、化学反応をおこした箇所は絶縁部になる。この性質を利用して PCBE で描いた回路をプリント基板に印刷するために版下印刷というものを行う。版下印刷をした場合、以下の図 4.11 のように印刷される。

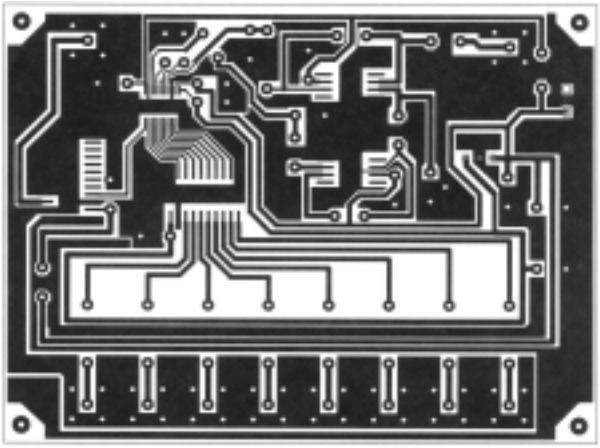


図 4.11: 版下印刷

つまり、導通する箇所は黒く、絶縁される部分は透明になる。前述の通り感光基板は光に反応する。絶縁部には光を当て、導通部には光を当てないようにしたいのでこの版下印刷は OHP シートに印刷する。より濃く、より精細に印刷するためにインクジェット方式のプリンタで印刷した。露光をこの OHP シート越しに行うことで黒く印刷されたところは化学反応せず導通部に、透明の部分は絶縁部になるというわけである。これによりパターンフィルムが完成する。

4.3.2 露光、現像、エッチング

パターンフィルムの作成が完了したら次に感光基板に露光を行う。感光基板は光に反応するが、最も反応しやすい波長は 360 nm の程度の光、つまり弱紫外線である。これを基板に露光させるのが最も時間的にも仕上りの具合としても効率が良い。そこで感光基板に露光させるための専用装置であるちびライト (サンハヤト社小型感光基板用露光ライトボックス BOX-1) を用いて露光した。露光時間は 300 秒ほどである。あまり露光しすぎると導通部にまで紫外線があたり絶縁部となってしまう。また逆に露光時間が不足すると絶縁しきれなくなり回路に不具合が生じるため注意が必要である。

露光が完了したらパターン現像作業を行う。露光処理工程で光の当たった感光被膜は現像液に容易に溶解するため、感光被膜を導通部のみ残すことが可能である。現像液にはサンハヤト社の現像剤 DP-10 を用いた。この現像剤は 40 度程度の湯に溶かして使用する。DP-10 は 200 ml の湯に溶かすことで A6 サイズの基板を約 3 枚現像することができる。現像時間は約 1 分であり、現像後は水で洗浄する。

現像後にエッチング作業を行う。エッチング作業により銅箔面を融解することができ、感光被膜でレジストされていない銅箔面はエッチング液に溶解されることになる。これに

よって銅箔面は感光被膜で守られた導通部のみになる。エッチング液はサンハヤト社のプリント基板用エッチング液 H-1000A を用いた。現像を行った基板を液温が 40 度ほどになるように湯煎をしたエッチング液に浸し、融解させる。エッチングは液の温度、疲労度によって融解度が激しく変化するので、表面の変化を目で確認しながらエッチングを行う。エッチング液は塩化第二鉄水溶液で pH 1 以上の強酸性液体のため、十分に注意しながら作業を行う。

エッチングによって絶縁部の銅箔面が剥離したら基板を水で洗浄する。この時点で基板は基板の台盤のみの絶縁部と、感光被膜に覆われた銅箔面の導通部に分かれている。感光被膜は完成した回路基板には不必要なものなのでエチルアルコールで剥離した。

エッチング処理が終了したら、あとは穴を空けて ADC ボードに必要な部品を半田付けする。これにより ADC ボードが完成する。

図 4.12、図 4.13 に出来上がった ADC ボードの写真、図 4.14 に接続を行ったセットアップ全体の写真を示す。

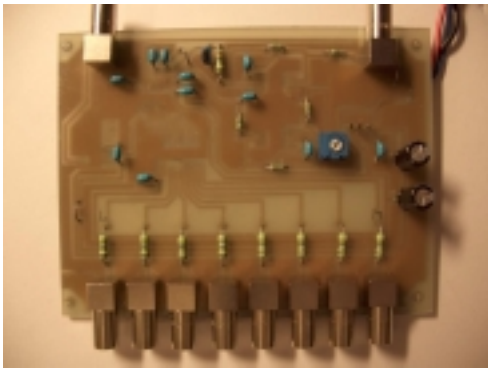


図 4.12: ADC ボード表

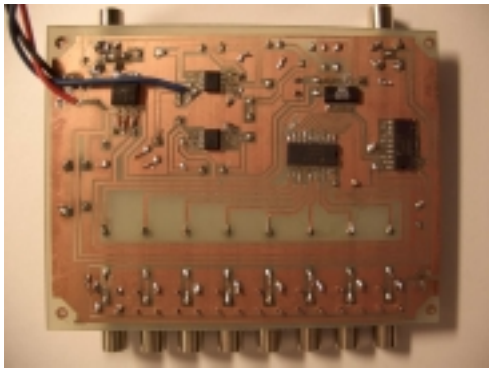


図 4.13: ADC ボード裏

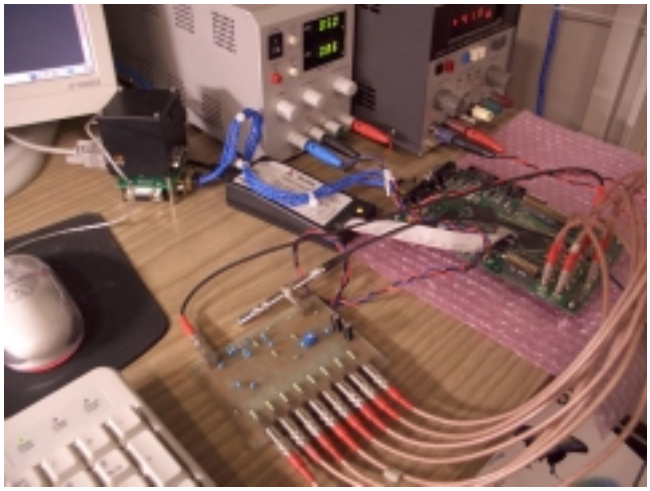


図 4.14: 全体セットアップ写真

4.3.3 ADC ボードの評価

製作した ADC ボードが設計通りにアナログ信号をコンバートしているかどうかをオシロスコープ、ファンクションジェネレータを用いて評価を行った。

サンプリングレートとなるクロックは NIM モジュールによって生成した。この NIM モジュールによって 40 MHz のクロックを ADC ボードに出力する。なお、FPGA と ADC ボードを接続し、ADC ボードのデジタル信号を FPGA に取り込む際は、FPGA で生成したクロックを ADC ボードに入力する。こうすることで任意のクロック信号を SpaceCube から SpaceWire を通じて変更することができる。また、FPGA と同じクロック信号を用いることで、サンプリングレートと FPGA 内処理クロックを揃えることができるため、入力データをラッチする際も、また FPGA 内で処理を行う際もデータが溢れることなくスムーズに処理を行うことが可能なためである。

アナログ入力信号は上記で述べたようにファンクションジェネレータ (Tektronix 社 AFG310) を用いた。ファンクションジェネレータというものは、その名前の通り特定の関数形の波形を出力するものである。代表的なものでサインカーブ、矩形波、三角波などである。Amplitude、Offset 電圧、Frequency なども自由に変化させることができるため、評価に用いるには適すると言える。

これらの機器を用いて ADC ボードがアナログ信号をデジタル信号へとコンバートできるのかどうかを確かめた。図 4.15 から図 4.22 が入力信号と出力信号のオシロスコープでの画像である。入力信号が黄色で ADS831 からの出力信号が青色である。

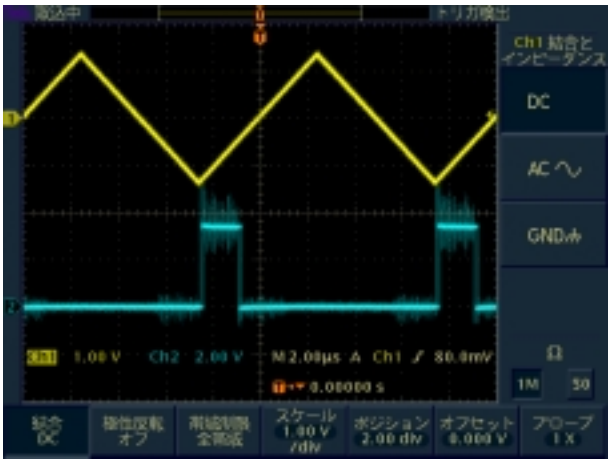


図 4.15: Bit1(MSB)

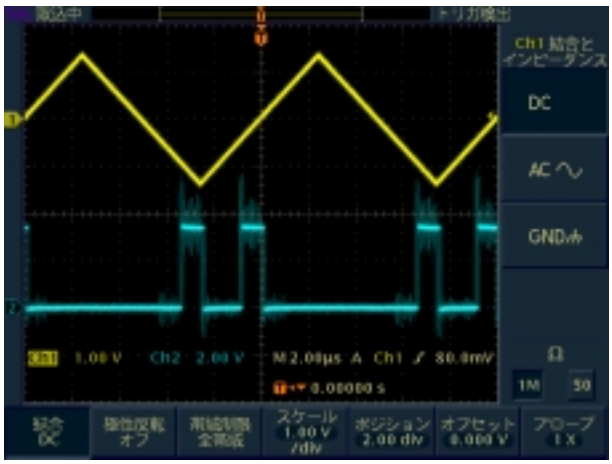


図 4.16: Bit2

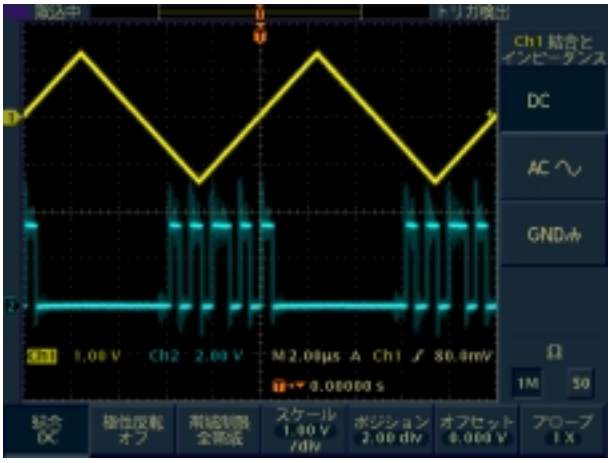


図 4.17: Bit3

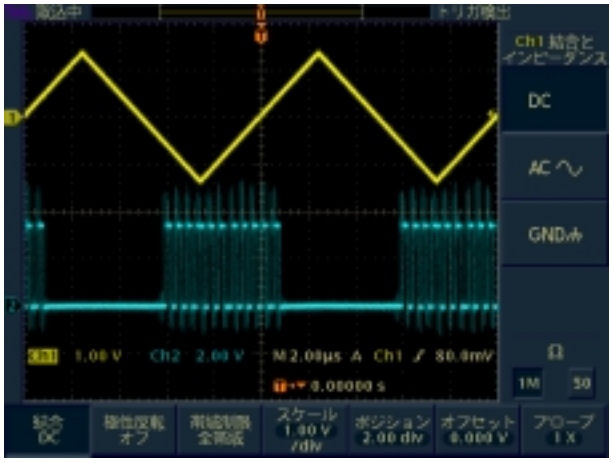


図 4.18: Bit4

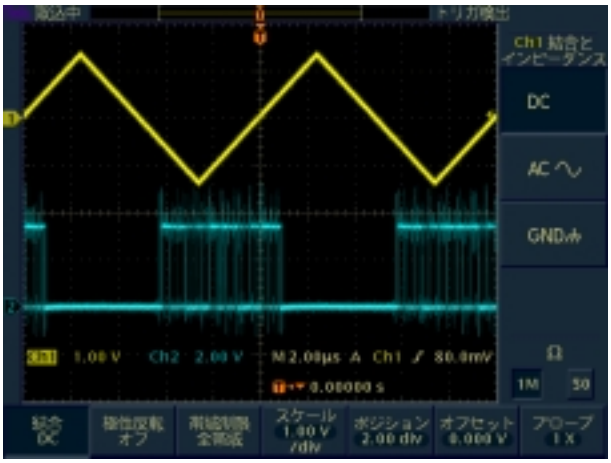


図 4.19: Bit5

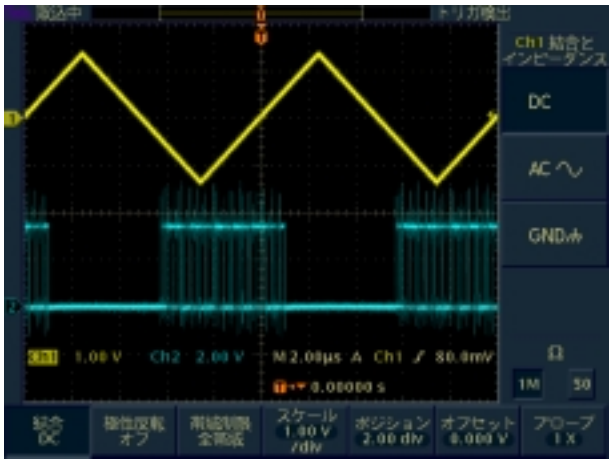


図 4.20: Bit6

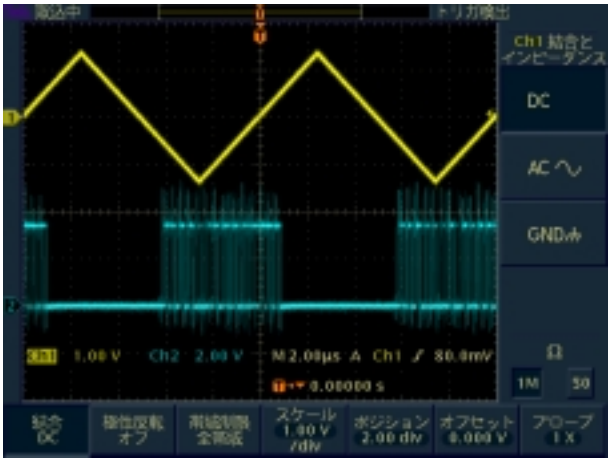


図 4.21: Bit7

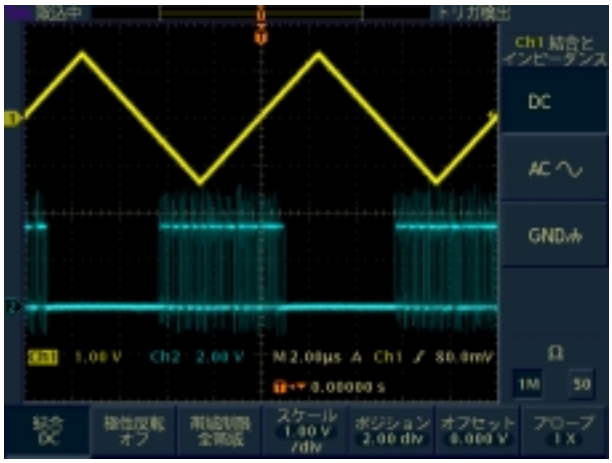


図 4.22: Bit8(LSB)

上位ビットである MSB(Most Significant Bit) はリファレンス電圧のトップ電圧とボトム電圧のちょうど間で High か Low が決まる。ADS831 ではトップ電圧が+3.0 V、ボトム電圧が+2.0 V なので MSB は+2.5 V を境に High か Low が決まる。Bit2 はさらにその半分、0.75 V を境に High か Low かを決定する。以下ビット数が増す毎に半分ずつとなっていく。具体的な例をあげると、例えばリファレンス電圧のトップ電圧が+3.0 V で、ボトム電圧が+2.0 V、かつ分解能が 4bit であるとする。そこへ電圧が+2.4 V の信号が入ってきたとする。MSB は+2.5 V を境に決定する。この場合入力電圧は+2.4 V なので MSB は Low、すなわち'0' を出力。Bit2 は MSB が Low だったのでその Low をより詳しくするために出力する。ボトム電圧は+2.0 V、MSB の境界は+2.5 V より、Bit2 は、入力信号がその半分の+2.25 V より大きいかどうかを判定する。この場合だと High、すなわち'1' を出力する。

あとは同様の操作を繰り返す。結果、MSB='0'、Bit2='1'、Bit3='1'、LSB='0' となる。LSB とは Least Significant Bit のことで、最下位ビットのことである。つまり+2.4V の入力信号はデジタル信号に直すと"0110" となる。なお、MSB より順次出力されるかのように書いたが、パイプライン型の ADC は変換が確定してから出力されるため、以上 4Bit は同時に出力される。

今回の出力画面では MSB は一つの矩形波しか出しておらず、ビット数が増えるにつれその矩形波の数は増している。矩形波の出力も目的に沿った場所、つまり MSB なら+2.5 V を境に'1' か'0' を出し、下位のビットになるにつれ、'1' と'0' の変動が多くなっている。これは ADC ボードの設計通りであるといえる。

これらからこの ADC ボードはアナログ入力信号をデジタル信号として出力しているといえる。さらに出力電圧は 0 V と+3.3 V の二つとなっているため、FPGA に入力するのに適している。

これで ADC ボードは完成したと言える。ただこのボードではただ入ってきた信号が ADC されて出力されているだけなので、入力信号がどのような信号だったのかということが分かるわけではない。ADC ボードの次の Digital I/O ボードでその処理を行う。

4.4 ADC ボードと SpaceWire Digital I/O ボードを用いた Oscillo System の製作

ADC ボードからのデジタル出力信号を Digital I/O ボードに入力したあとは I/O ボードでの User FPGA で処理を行う。このための UserFPGA 内の回路は前章で述べたハードウェア記述言語、VHDL を用いて回路を書いた。UserFPGA 内でユーザが具体的な処理を行う部分を User Module と呼ぶが、UserFPGA の User Module の新規開発、UserFPGA と SpaceWire FPGA とのバス接続、などは東京大学牧島研湯浅氏が作った UserFPGA の VHDL のテンプレートがあるのでそれを用いた。このテンプレートには RMAP のメモリ配置を決める Address マップライブラリ、ピンアサインを設定するユーザ制約ファイルも含まれるため、UserFPGA の機能を設定する User Module の部分を製作することのみに集中することができる。図 4.23 に UserFPGA の簡単なブロック図を示す。

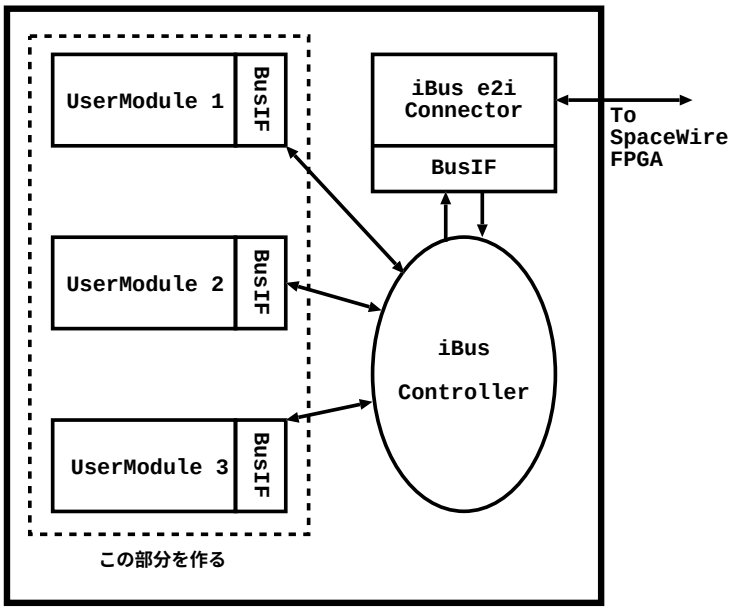


図 4.23: UserFPGA 内での簡単なブロック図

SpaceCube と直接やりとりをするのは SpaceWireFPGA であり、ユーザが介入することの出来る UserFPGA からは SpaceWire と間接的にしかつながっていない。その SpaceWireFPGA と処理命令をやりとりするのが iBus e2i Connector(Internal Bus External to Internal Connector) である。また、User Module は複数作成することができるが各 User Module を管理し iBus e2i Connector からの処理を所定の Module へと転送するためとして iBusController(internal Bus Controller) がある。つまり iBusContoroller にはハブとしての役割がある。更に User Module はこの BusController との通信を行うために各 User Module に BusIF(Bus InterFace) というものをインスタンス化している。

4.4.1 UserFPGA 内での目的

今回はデジタル信号を取得し、その波形をオシロスコープのように再現することを目的とした。そのためとして二つの User Module を製作した。一つが Trigger Module、もう一つが Oscillo Module である。図 4.24 に UserFPGA 内部 User Module での信号処理のブロック図を示す。

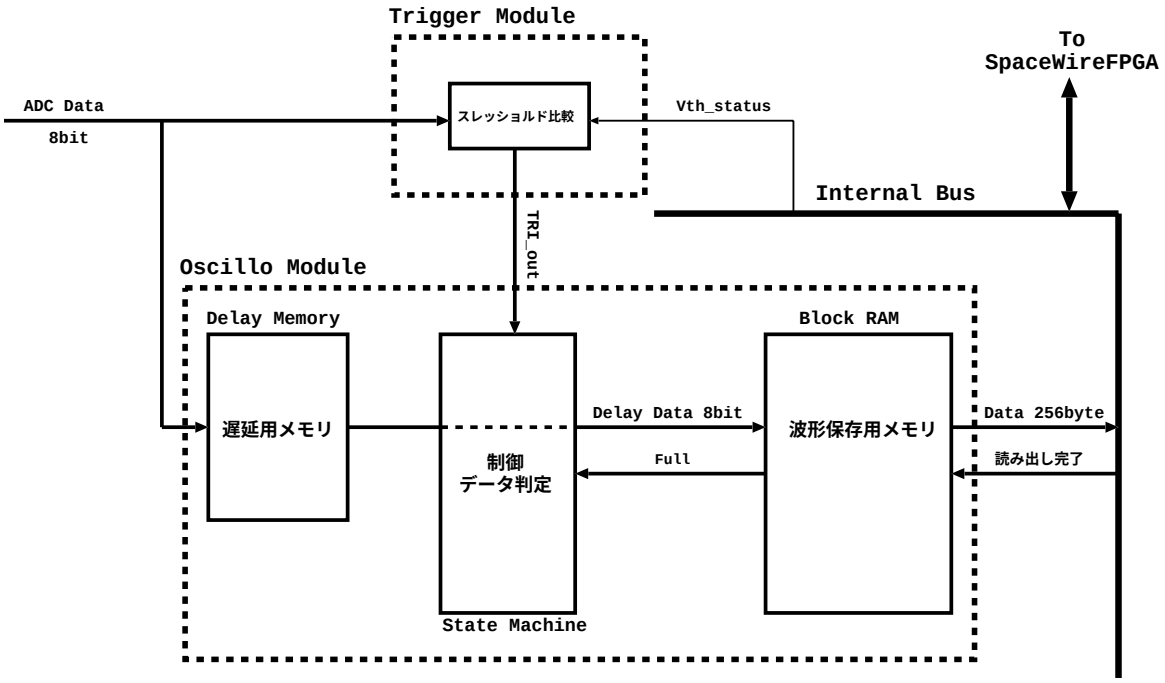


図 4.24: User Module ブロック図

4.4.2 Trigger Module

Trigger Module は ADC ボードからの信号とユーザが決めた値とを順次比較していき、ユーザの設定値より入力信号が高ければ Trigger 信号として'1' を出力するためのモジュールである。これは ADC ボードからの信号は常に有用な信号であるということはなく、大多数が意味のない信号ゆえ、それらを選別するためにある。Trigger 信号が'1' になったときのみの信号を処理することで FPGA 内での無駄な処理を抑えることができる。また、こうすることで最終的に PC に入るデータの量を減らすことができ、整理もしやすくなる。図 4.25 に Trigger Module の状態の遷移図を示す。

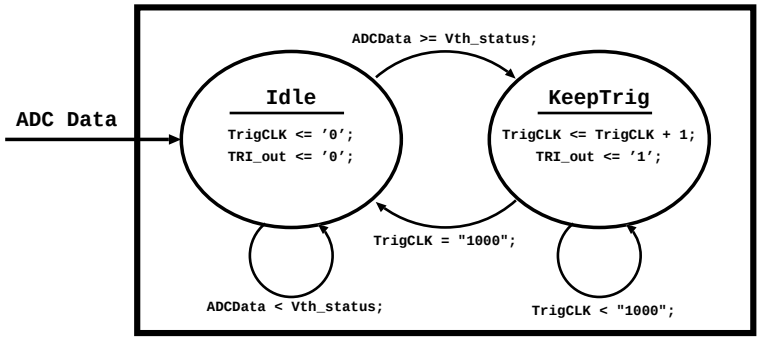


図 4.25: Trigger Module 状態図

以下に簡単に状態と信号の説明を行う。

状態 Idle
ADC ボードから信号が入ってきたときにこの Idle 状態で Threshold 電圧比較を行う。Threshold 電圧は RMAP によって SpaceCube から変化させることができ、0 V から Digital I/O ボードの最大入力電圧である+3.3 V までの範囲を 256 分割した中から選ぶことができるようになっている。その Threshold 電圧は FPGA の中では 16 Bit の信号で表示されている。ここではその信号を Vth_status とする。この Vth_status より ADC ボードからの信号が大きければ次の KeepTrig 状態へ遷移する。

状態 KeepTrig
ADC ボード信号が Vth_status を上回っていれば、この ADC ボード信号が有用であることを Oscillo Module へ伝えるために TRI_out 信号というものを'1' にして伝える。しかし、ADC ボードの信号が Vth_status を一瞬しか上回らなかった場合、極端な話として 1 Clock の間しか上回らなかった場合だとすぐに TRI_out 信号が'0' となる。1 Clock では次の Osicllo Module の処理は終了していないため、TRI_out 信号が急に'0' になることは Osicllo Module が誤作動を起こしかねない。それを防ぐために一度 TRI_out 信号が'1' になれば、しばらく TRI_out 信号が'1' を出力するように設計しなくてはならない。この KeepTrig 状態でそれを行うために、新たに TrigCLK という信号を用意し、クロックカウンタとして用いる。KeepTrig 状態に遷移して、8 Clock 分カウントするまでは Idle 状態に戻れないようにすることで、Oscillo Module が誤作動を起こさないようにする。8 Clock の間に新たな有用な信号が来ることもあるかもしれないが、一度 TRI_out が'1' になれば次の Oscillo Module で少なくとも 256 Clock かけて波形を保存するプロセスがあり、波形保存が完了しないと TRI_out を再び受け付ける状態にならないため、8 Clock という長さは適切である。

TrigCLK が 8 Clock カウントしたら最初の Idle 状態に移り、TrigCLK をリセットする。これで Trigger Module は ADC データが Vth_status を上回れば少なくとも 8 Clock は TRI_out を'1' を出力する。

4.4.3 Oscillo Module

Oscillo Module は、FPGA に入力された波形信号を Trigger Module で作成された TRI_out 信号を用いて有用かどうかを判定し、有用ならば波形信号を保存し、保存されたデータを SpaceCube へと渡すモジュールである。取得したデータを SpaceCube を通して Linux 等の PC で解析を行うことができればどのような波形だったのかという再現等など様々な処理が行える。

波形を保存する場合、TRI_out 信号が'1' になったときをスタータとして保存を始めるが、TRI_out 信号が'1' のときのみを保存するのでは、波形が変化を始めた瞬間のデータ

を入手することができない。変化がある程度進み、Trigger Module が内部で設定したスレッショルド電圧より波形信号が高くなったときに初めて TRI_out 信号を'1' にするからである。この問題を解消するため、Oscillo Module 内で入力してきた信号を遅延させるためのメモリを設置した。このメモリは入ってきた信号を一時的に常に書き込む。その書き込みを行ったときのメモリアドレスを書き込みアドレス、読み込みを行ったときのメモリアドレスを読み込みアドレスとすると、このメモリでは書き込みアドレス値より一定値少ない値を読み込みアドレスとして外部へと出力する。つまり、その一定値を 4 とすれば、書き込みアドレスが 10 なら読み込みアドレスを 6、書き込みアドレスが 20 なら読み込みアドレスを 16 とするということである。こうすることでメモリの出力信号は入力信号より時間的に遅れた信号が出力されるというわけである。図 4.26 には書き込みアドレスとして addrW、読み込みアドレスとして addrR を準備した。メモリにはこのアドレスで指定したところに書き込み、読み込みを行うようになっている。図 4.26 で同じ時刻でも書き込みと読み込みがずれていることから ADC ボードからの信号が遅れて出ていることが分かる。

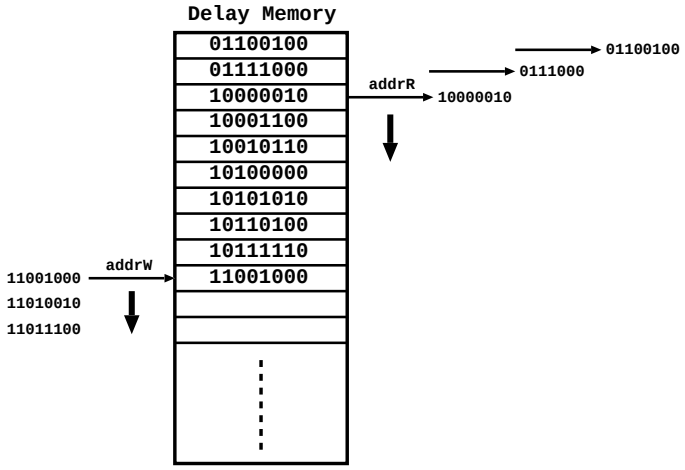


図 4.26: 遅延用メモリイメージ図

しかしこの方法ではメモリを起動させた瞬間に出力される信号はまだ入力がされていないアドレスからの出力となり不定信号として出力されてしまうため、遅延用メモリの delay_data 信号が有用となるまで待機する必要がある。そこで delay_data 信号が有用であることを示すために新たに UseDelayData という信号を作る。この信号が'1' になれば遅延用メモリから出力されたデータ信号が使えるということを示すフラグ信号である。この UseDelayData 信号が'1' になるのは addrW で入力した位置のデータを addrR で出力するときである。UseDelayData 信号と遅延用メモリとの関係イメージを図 4.27 に示す。

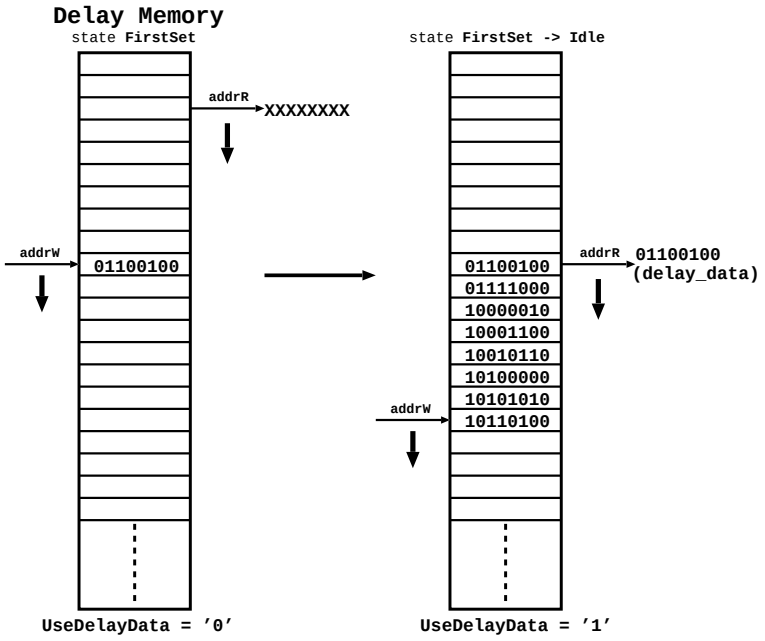


図 4.27: UseDelayData 信号

図 4.24 にあったように、ADC ボードから入力された 8bit 信号は Trigger Module と Oscillo Module と別々に入力されるようにする。こうすることで Trigger Module は Oscillo Module で遅延された信号とは全く関係なく、リアルタイムで入ってきた波形信号のスレッショルド判定を行うことが出来る。この Trigger Module からの出力信号である TRI_out='1' 信号が Oscillo Module に入ってきたとき、Oscillo Module では遅延のために TRI_out 信号が'1' になった元である波形もまだその変化を見せる前のため、波形を最初から保存することができるというわけである。今回、その遅延は 24 MHz でサンプリングをしたとして、64 Clock 分、すなわち約 2.7 us になるようにした。この 64 Clock という値を timerug_status として信号としている。

遅延をした入力信号、ここでその信号を delay_date とすると、delay_data は TRI_out 信号が'1' となった波形として、波形保存用の RAM へと書き込まれる。この RAM は予め領域が決まっており、一度書き込みが始まれば領域全てに波形を保存するまで書き込みを行う。最後まで書き込みを行った後、保存された波形を SpaceCube へ転送可能であることを示すフラグとして読出準備完了信号 ReadableFlag 信号を'1' にする。また、RAM に保存したデータはそのまま放っておくと次の TRI_out 信号が'1' になったときに新たに上書きされてしまうことになる。これを防ぐために SpaceWire 経由で SpaceCube から値を読み込まれるまで、いくら TRI_out 信号が'1' になっても上書きをしないようにした。これを解除するには SpaceCube 側から SpaceWireFPGA のアドレス空間に設定した解除用メモリにデータ読出完了を合図する必要がある。この時の信号を ReadFinishFlag 信号とする。この信号が'1' になれば次の波形を得る準備を行うことができるというわけである。

以上を考慮し、次に示すような流れで RAM へ書き込みを行う。

RAM への書き込みプロセス

1. delay_data 信号が有用になるまで待つ
2. Trigger Module からの TRI_out 信号が'1' になるのを待つ
3. TRI_out 信号が'1' になったら RAM に書き込みを開始する
4. RAM への書き込みが完了したら ReadableFlag 信号を'1' にする
5. SpaceCube から読み出しが終わったら ReadFinishFlag 信号を'1' にする
6. ReadableFlag 信号、ReadFinishFlag 信号を'0' にし、2. に戻る

図 4.28 に作成した部分の詳細なブロック図を、図 4.29 に Oscillo Module における状態遷移図を示す。状態遷移図とは、ステート・マシンの状態が遷移するための条件や状態名をしめした図である。通常、回路は入力信号によって出力信号が決まるが、このような、入力信号だけではなく、入力信号を受け付ける際の状態がどうなのか、そして決められた手順によって出力信号が決まる回路のことをステート・マシンと呼ぶ。今実験のように、書き込みプロセスで TRI_out 信号が入力されるまで待機しておく状態、実際に書き込みを行う状態、SpaceCube から読み出しが完了されるまで待機しておく状態などのように手順を踏まえての回路を作成するのであればステート・マシンを用いるのがよい。

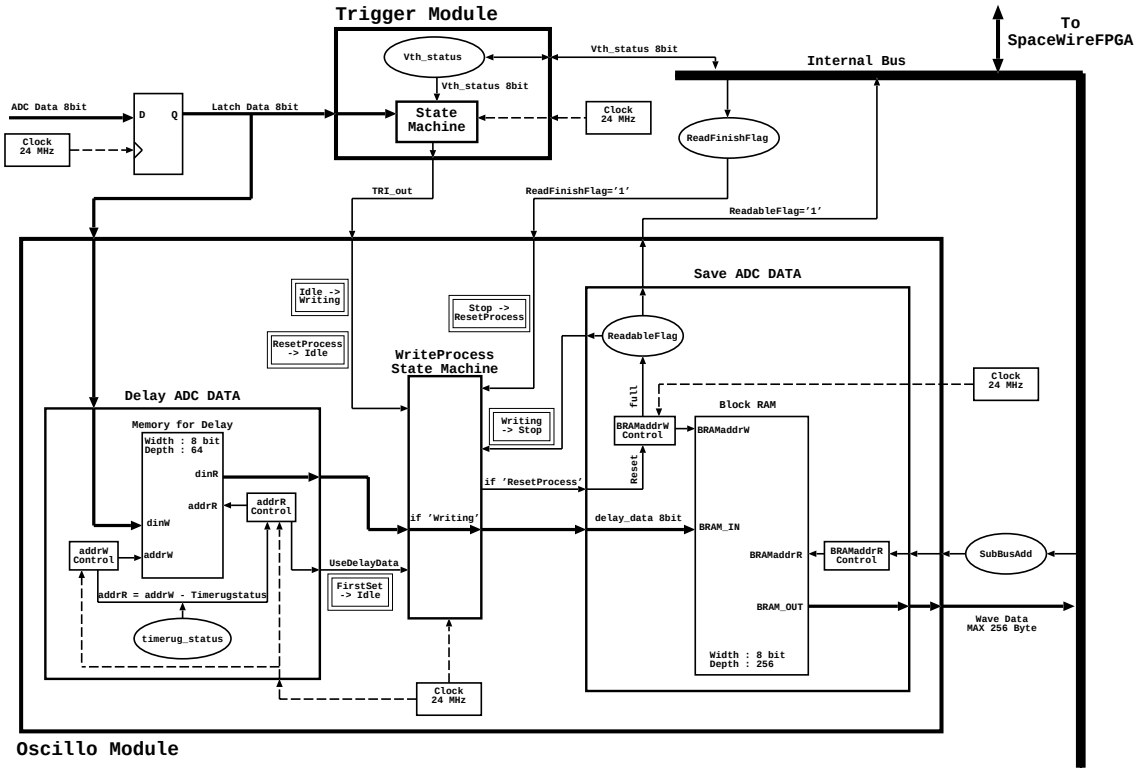


図 4.28: 詳細な User Module ブロック図

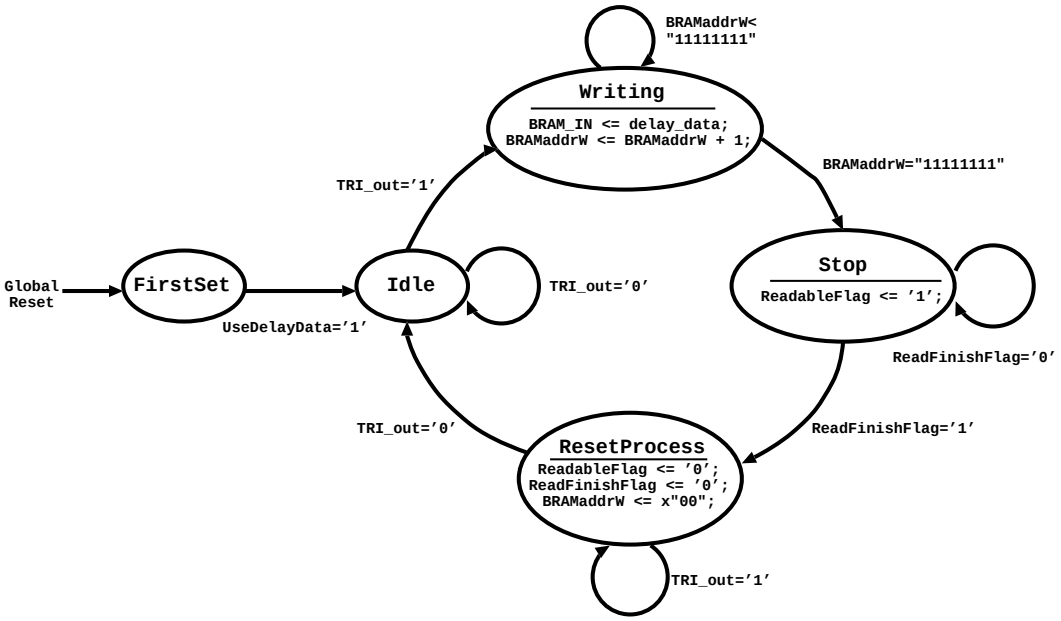


図 4.29: Oscillo Module 状態遷移図

状態 FirstSet
RAM **への書き込みプロセス**:1
delay_data が有用であるかどうかを示すフラグ信号 UseDelayData が'1' になれば次の状態 Idle へと遷移する。この FirstSet 状態には全機能をリセットする GlobalReset 信号がオンにならない限り戻ってこない。

状態 Idle
RAM **への書き込みプロセス**:2
この状態は Trigger Module からの TRI_out 信号によって次の状態へと遷移する。それ以外の情報では遷移しない。TRI_out 信号が'1' になったら遅延用メモリから出力される delay_out 信号を RAM に書き込まないといけないため、次の Writing 状態へと遷移する。

状態 Writing
RAM **への書き込みプロセス**:3
この状態では遅延用メモリか内の波形信号を Block RAM へと書き込んでいき、書き込んだ後に Stop 状態へと移る。RAM へと書き込む際にはどこに書き込むのか、SpaceCube から読み込むときにはどこを読み込むのかといった RAM のアドレスを指定する必要がある。そのためにアドレスを指定する Signal である、BRAMaddrW、BRAMaddrR の二つの Signal を準備しておく。BRAMaddrW は入力するアドレス値を格納、BRAMaddrR は出力のそれを保存しておく。BRAMaddrW はカウンタのように、あるアドレスに書き込みを行ったらインクリメントして次のアドレスへと移動する。今回、波形データを保存するための RAM のサイズとして、1 Clock で保存する 8 bit の領域を 1 ブロックとすると、全部で 256 ブロック準備した。8 bit が 1 Byte なので 1 Clock で 1 Byte、すなわち総 RAM ブロックサイズは 256 Byte である。24 MHz でサンプリングした場合 10.7 us 分のデータを保存することが出来る。以下の図 4.30 に RAM への書き込み過程のイメージ図を載せる。

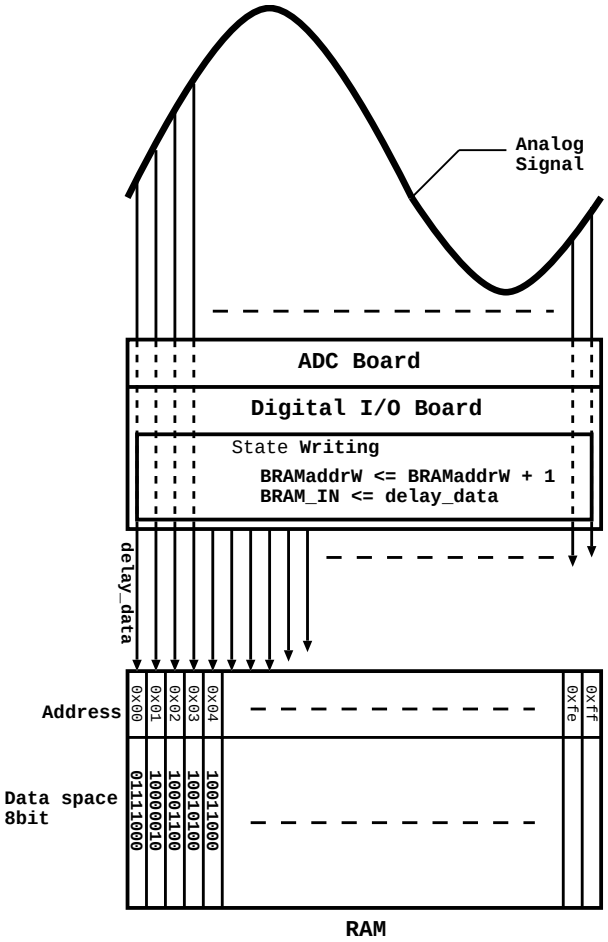


図 4.30: RAM への書き込み過程イメージ

BRAMAddrW は書き込み位置を決めるための Signal で、図 4.30 にあるように RAM のどこへ書き込めばいいかを指定する。効率よく RAM に書き込むために Writing 状態の間はカウンタのように RAM の最下位アドレスから一つずつ書き込むようになっている。この RAM には 256 ブロックのデータを格納する領域が用意されているが、BRAMAddrW は 256 番目のブロックに delay_data を入力するまで動くようになっている。つまり、一度 TRI_out が'1' になり、Writing に状態が遷移したら RAM に保存されるデータサイズは常に 256 Byte になるということである。TRI_out 信号の長さにもよらない。

TRI_out の長さに依存しないで一定のデータ量を得られることでデータの整理を行いやすく、更に回路がシンプルにまとめることができ、更に後ろに尾を引くような波形であっても TRI_out 信号に依存しないことからその波形を保存をすることができる。

最後まで書き込んだ後は Stop 状態に移っているため RAM のデータが SpaceCube から読み出されない限り、TRI_out 信号が再び'1' になっても RAM に書き込まれることはな

い。これにより確実にデータを保存することが可能である。

BRAMaddrR は UserFPGA では値を指定しない。この Signal は RMAP により、SpaceCube 側から SubBusAdd 信号として読み出しをするアドレスを指定されたとき、そのアドレスを RAM 用に変換されたあとの信号を格納するためにある。例えば、図 4.30 の場合、SpaceCube がアドレス 0x02 のデータを読み取りたいと言った信号を Digital I/O ボードに送ったとき、そのアドレス値を BRAMaddrR に渡し、その値”10001100” を出力するという仕組みである。そのため、SpaceCube 以外では BRAMaddrR は変化させることはできない。

状態 Stop
RAM **への書き込みプロセス**:4
状態 Stop は RAM への書き込みが完了し、その後 SpaceCube から読み出しが完了するまでの間の待機状態を示す。RAM へ上書きをされるのを防ぐために書き込みが出来ないようにしていてもいつまでも停止しておくわけにはいかない。ここでは SpaceCube から読み出しが完了したという信号が SpaceWire FPGA の指定メモリに書き込まれたら ReadFinishFlag が’1’ になり、すぐに次の状態へと遷移するようにする。

状態 Reset
RAM **への書き込みプロセス**:5**、**6
状態 Reset では新しい信号が来たときに先ほどと同様の処理が出来るように初期化をしておく工程を行う。具体的には書き込みアドレス BRAMaddrW、SpaceCube へのデータ読出準備完了信号 ReadableFlag、SpaceCube からのデータ読出完了信号 ReadFinishFlag の各信号の初期化である。さらに重要なのが TRI_out 信号の確認である。次の状態である Idle に遷移したと同時に TRI_out 信号が’1’ であるためにすぐに処理を行った場合誤作動を起こしかねない。そこで、TRI_out 信号が’1’ の場合は’0’ になるまで待機する。TRI_out 信号が’0’ になれば最初の Idle 状態へと遷移する。

以上の過程をタイミングチャートで示したのが次の図 4.31 である。

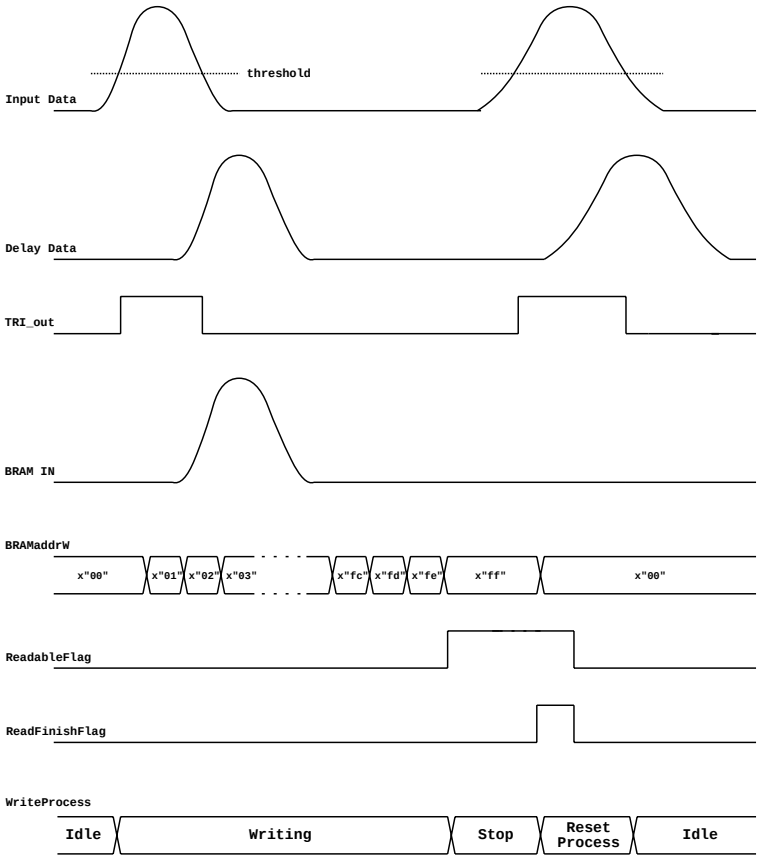


図 4.31: Oscillo Module タイミングチャート

4.5 SpaceCubeの立ち上げ

Digital I/O ボードのデータ信号を読み取るためには SpaceWire 規格を実装した SpaceCube が必須である。ここではその入手から実際に使うまでの過程を述べる。

SpaceCube は、キーボード、マウス、ディスプレイを接続することで普通の PC と同様の操作をすることができるが、SpaceCube で動作させるプログラムは Linux で開発を行う方が扱いやすいため、LinuxPC をホスト PC としてシリアルケーブルで SpaceCube とつないで SpaceCube をシリアルコンソール化をして動かすこととする。シリアルコンソールというのは、シリアルポート、シリアルケーブル経由で別の PC 端末から遠隔的に操作を行うことができる機能のことである。すなわちキーボード、ディスプレイなどといった PC の I/O 入力がシリアルケーブル経由になるということである。この設定を行うことでプログラムの開発、SpaceWire 経由での Digital I/O ボードへの RMAP を実質的にホスト PC である Linux で全て行うことが出来る。ホスト PC の Linux ディストリビューションは Vine 4.1、SpaceCube との接続にはシリアルケーブル RS-232 9 ピンクロスケーブル

を用いている。

次にOSだが、SpaceCubeは2.2節で述べたようにOSとしてLinuxかT-Engineかを選ぶことができる。本研究は将来的に宇宙で観測、処理、通信を行うという前提での基礎実験なのでリアルタイム性に優れるT-EngineをOSとして採用した。OSのKernelであるT-Kernel1.0.00はSpaceCube内蔵のFlashROMにシステムを書き込んだ。SpaceCube用のFlashROMイメージはSpaceCube付属CD-ROMにある。SpaceCube側からこのファイルをシリアルケーブル経由でダウンロードすることが出来る。また、SpaceCubeにはHDDがないため、CF(Compact Flash)の1GBを使用する。T-Kernelが使用できるようになったらインターネットに繋がるように設定をする。こうすることでシリアルケーブル以上の高速、複雑なデータ送信システムを組み立てることが出来る。本研究室ではLAN(Local Area Network)が構築されているが、SpaceCubeも他のLinuxPC同様の設定でインターネットに繋ぐことが出来た。

SpaceCubeとDigital I/OボードとはSpaceWireで接続する。SpaceWireケーブルはSpaceCubeには付属していなかったため、SpaceWire企画書 [6] に記していた設計図に基づいてD-sub 9ピンと、単芯ケーブルを寄り合わせてツイストペアにしたケーブルから作成したもので代用とした。

以上をまとめると図4.32のようになる。SpaceWire端子はSpaceCubeに直接実装されているわけではなく、図のようにフラットケーブルでつながった分配基板にある。図2.4のSpaceCube真下にあるボードがその分配基板である。

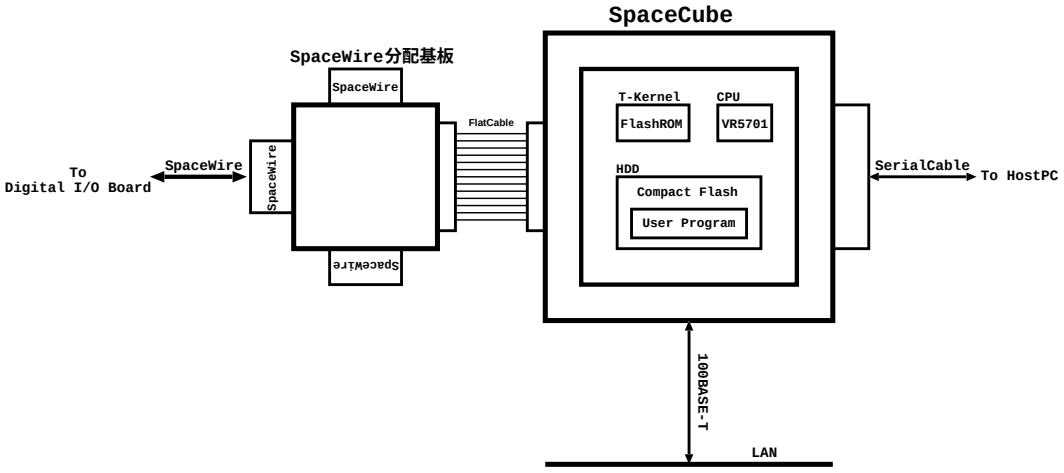


図 4.32: SpaceCube信号処理ブロック図

4.5.1 SpaceCube用プログラム

SpaceWireを経由してDigital I/Oボードと通信し、ボードのメモリのRead、Writeを行うためのSpaceCube用プログラムはホストPCであるLinuxPCで作成する。作成した

プログラムはLinuxPCでコンパイルしてシリアルケーブル経由でSpaceCubeへと送信する。こうすることでSpaceCubeでのプログラム作成も普段使っているLinuxPCで簡単に作ることが出来る。

しかし、研究に用いたLinuxPCのCPUはx86系の、SpaceCubeのCPUはMIPS系の命令セットを持ち、互いに異なるコンピュータ・アーキテクチャのため、LinuxPCでプログラムを作成、コンパイルして実行形式にし、それをSpaceCubeへと送信してもSpaceCubeでは実行できない。つまり、開発をLinuxで行うならばコンパイルをMIPSに合わせて行わなければならなくなる。そこでクロスコンパイルという技術を用いる。クロスコンパイルは開発を行っているPCとは違うコンピュータ・アーキテクチャに対応するようにコンパイルする技術である。これによってSpaceCubeへのプログラム作成をLinuxで容易に行うことが出来る。

SpaceCube付属CD-ROMにMIPS対応のコンパイラであるgccmips3.3.2とMakefile標準ファイルなどが内蔵されているため、ホストPCにダウンロードして使用する。それらをうまく使用するためにMakefileを作成し、さながら普段用いているgccを使うかのように設定した。

SpaceCubeがSpaceWire経由でDigital I/Oボードとの接続を確保しているが知るために大阪大学の能町教授が作成したプログラムを用いた。このプログラムによってSpaceCubeは正常にDigital I/Oボードと接続できていることが確認できた。

SpaceCubeがSpaceWire経由でDigitai I/Oボードと、シリアルケーブル経由でホストPCと接続することで実験の環境が整ったといえる。ここで図4.33に実験セットアップ全体のブロック図を示す。

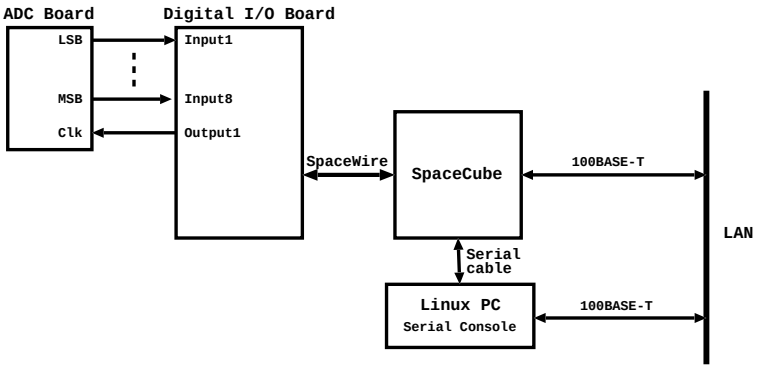


図 4.33: 全体ブロック図

4.5.2 SpaceCubeでのプログラム作成

SpaceCubeにはRMAPを用いてのDigital I/OボードのSpaceWireFPGAメモリ空間データのRead/Writeを行うプログラムと、収集したデータをLAN経由で解析用の

LinuxPC に送信するためのプログラムの二つが必要である。後者は効率を無視すればT-Kernel システムコマンドで ftp を行うことができるので時間と手間をかければプログラムを作らなくてもよいが、前者はこの方法でしか Digital I/O ボードのメモリ空間の値は読むことができない。

4.5.3 SpaceWireFPGA との通信プログラム

SpaceWireFPGA メモリ空間を読み取るために、能町教授の作成された C++言語ファイル、ライブラリを参考に新たにプログラムを作成した。このプログラムは UserFPGA の Oscillo Module で保存した RAM の値を読み取ることを主目的としているが、他にステータスチェック、Threshold 電圧変更のためのプログラムも同様に作った。それらをバラバラに作成するのでは効率が悪いので SpaceCube ではそれらを数字で選択し、指定のプログラムを走らせることができるようにした。以下が作成したその各プログラムのリストである。

- Read on Display
- Read and Send(TCPIP)
- Write Status
- Check Status
- Read single event
- Reset Block RAM

Read on Display
UserFPGA の Oscillo Module で保存した波形の値を確認するためのプログラムである。このプログラムを走らせると Block RAM の 256 ブロック全てを端末に表示する。波形を保存できているか確認を行うために表示する。

Read and Send(TCPIP)
最終的な目標は UserFPGA で処理した波形の値を LAN で繋がった LinuxPC で処理を行うことである。この Read and Send(TCPIP) では LAN の TCP/IP を利用して LAN に接続された PC へと波形の値を送信する。詳しくは 4.5.4 節にて述べる。

Write Status
ここでは FPGA で処理を行う時にステータスが変化すると処理結果が変わるであろうも

のを変化させるための書き込みを行う。今回は Threshold 電圧のみ変化させるようにした。Threshold 電圧は 256 通り変化させることができる。最大値は 256、すなわち 0xFF で+3.3 V となるようにセットした。最小値は 0x00 で 0 V である。

Check Status

今回 Oscillo Module と Trigger Module では様々な信号を用いている。あまりに多く信号を使っていると、User Module にエラーが発生したときにどの信号でエラーが発生したのか分からないのでは困るため、それらの異常を診断するためのプログラムを作った。それがこの Check Status である。ここでは各信号に対して Read を行い、返って来た信号と正常に動作していればこうなるであろうという予測値と比較して今この信号は正常かどうかを判断する。判断する信号と、異常か正常かを簡単に示したのが表 4.1 である。

Signal	正常
ReadableFlag	'1'
ReadFinishFlag	'0'
UseDelayData	'1'
timerug_status	0x40
WriteProcess	Stop(=0x0100)
Vth_status	(現在の状態表示)

表 4.1: Check Status の正誤判定

これ以外の数字であればプログラムはエラーを返す。

Read Single event

各信号は SpaceWireFPGA のメモリアドレスを指定することで読み取りを行うことが出来る。ここではそのアドレスを指定し、その値を返すようにプログラムを作成した。Check Status の表 4.1 の信号は勿論、波形保存用 Block RAM の値も一つずつ読むことができるようにしている。

Reset Block RAM

例えば ADC ボードの入力信号源が変化した、Threshold 電圧を変化した等、今 Block RAM に保存している波形値をクリアし、新しい波形を保存したい場合などにこのプログラムを用いる。ここでは ReadFinishFlag を'1' にするように書き込みを行う。こうすることで UserFPGA の Oscillo Module は SpaceCube が読み取りを完了したとし、次の ResetProcess 状態に移るため新たな波形を Block RAM に保存するようになる。

4.5.4 LinuxPC との通信プログラム

SpaceCube で Digital I/O ボードからの波形を取得したらそのデータを LAN に転送する。CF カードに保存することもできるが CF カードは 1 GB の容量しかなく、データ収集を重ねればすぐに容量が足りなくなる。また解析を行うのであればやはり SpaceCube にデータを残しておくのではなく、LinuxPC へと転送する方が効率的である。

通信手段は様々ある。T-Kernel のシステムコマンドとして ftp コマンドがあるため、LAN に繋がっていることを利用し、ftp サーバにファイルを転送、解析用 LinuxPC にダウンロードするということも可能である。しかしこの方法では、得たデータをいちいち SpaceCube でなんらかのファイル、例えば dat などに保存しなくてはならない、ftp サーバに一度保存しなくてはならないといった手間がかかる。一つの波形だけを送信するだけならまだ良いが、一度に多量のファイルを送信することになればこの手間も無視できなくなる。そこで Socket 通信というものを行う。

Socket 通信というのはアプリケーションのプロセス間における通信手段のことで、TCP/IP を利用しての通信手段では必ずといっていいほど用いる。

Socket 通信では LinuxPC をサーバに、SpaceCube をクライアントにして通信を行う。LinuxPC をサーバ化させるのはデータを受信する上で効率がよいからである。SpaceCube は LinuxPC の IP アドレス、サーバの開放ポート番号が分かっていれば Socket 用関数を用いることで簡単にデータを送信することができる。今回は手動で送信を行うため、複雑なプロセスは行わなず、一つずつ送信するようにした。

送信データは通常様々な情報を圧縮したパケットを作って送信する。これを受け取った PC が情報の再構築を行い、そのデータがどのようなものであったかを知る。今回はそのような制御情報としてデータサイズを付加した。これは Socket 関数の送信にあたる send 関数、受信である recv 関数には実際に送信するデータとそのデータサイズを指定する必要があるためである。このサイズを各々の関数が指定しても良いが互いに異なるサイズを指定するとエラーが発生してしまう。そこで送信側が予めデータサイズを計算し、その計算値をパケットに付加して送信する。受信側はそのパケットの指定したデータサイズ値が書かれた箇所を読み取り、受信データのデータサイズを知る。こうすることで異なるデータサイズであったとしてもエラーを起こしにくくなる。

以上の過程を踏まえると、SpaceWire Digital I/O ボード、SpaceCube の制御を全て一つのパソコンから行うことができ、かつそのデータ処理をも行うことができる。

4.6 製作した Oscillo System の評価

この Oscillo System が入力信号をきちんと再現できているかの評価を行う。しかしながら、ADC ボードの ADS831 は RSEL が'H' の状態のとき+1.5 V から+3.5 V の間の電圧

を変換し、その範囲に収まるためにADS831に入力する前に調整をするのだが、調整用OP641が壊れていたため、入力電圧が若干下寄りになっている。そのため入力信号の下半分が切れているという状況になっているが、本研究の本質はADCが思った通りに働くようにすることではなく、あくまでADCされた信号が元の信号と同じようにFPGAを用いて再現できるかどうかということなのでここではそのままのADCボードを用いて実験を行った。

4.6.1 入力電圧関数の違い

用いたFunction Generatorはその名前の通り出力電圧を関数に例えて出力することができる。ここではそれらを関数別に表示する。なお、一度だけではそれらが正しく再現できているか分からないため4回同じデータ収集を行い、それらを比べる事で再現性を吟味する。なお、全ての関数について共通してFPGAでのThreshold電圧は約0.0129V、つまりVth_statusを0x01にしている。これは関数が入ればすぐに調べたいからである。なお、入力電圧ノイズの影響に関しては、先ほど述べたようにADCボードの変換可能入力範囲はアナログ入力端子に入る電圧よりかなり上側にシフトしているため、ノイズ部はADS831の入力部には入らない。つまりノイズは常にADCが'00000000'と変換するため、ノイズによってTRI_out信号が立つことはない。

SIN

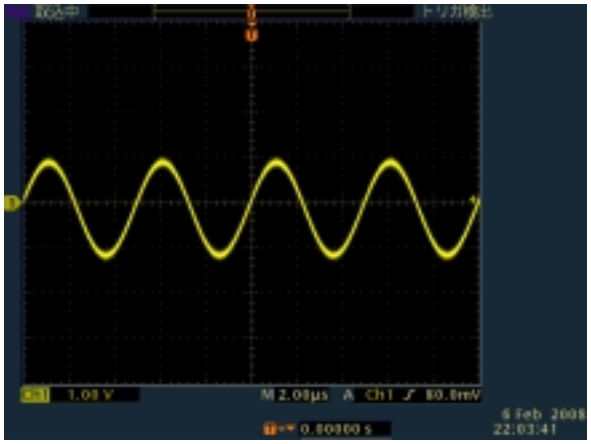
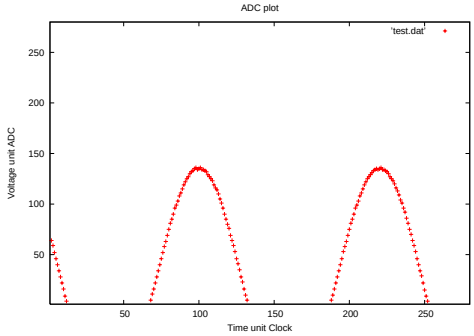


図 4.34: SIN 関数のアナログ入力電圧

Status	value
Amplitude	2.0 V
Frequency	200 kHz

表 4.2: SIN Status



TRIA

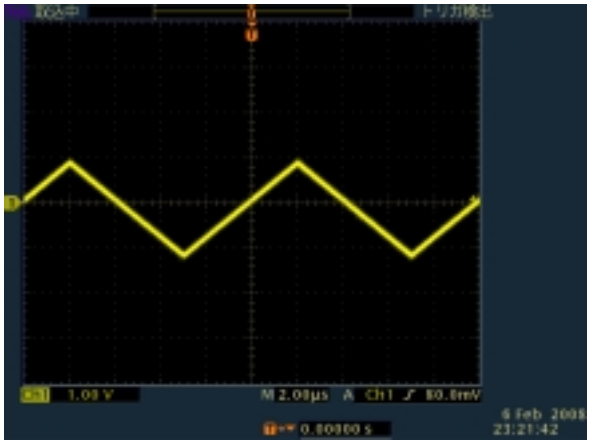
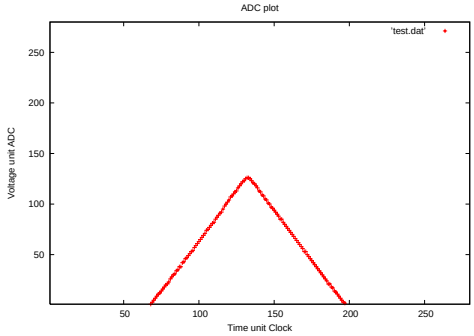


図 4.39: TRIA 関数のアナログ入力電圧

Status	value
Amplitude	2.0 V
Frequency	100 kHz

表 4.3: TRIA Status



RAMP

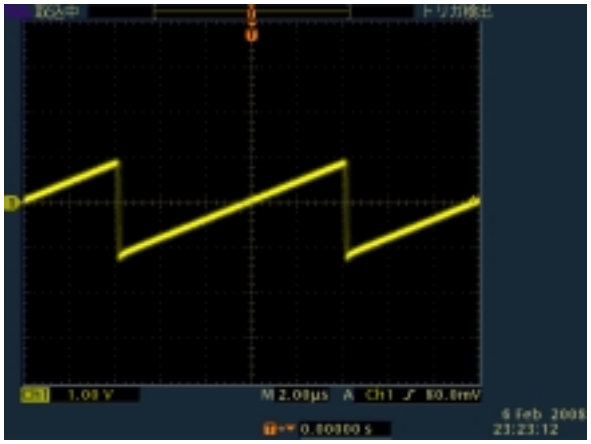
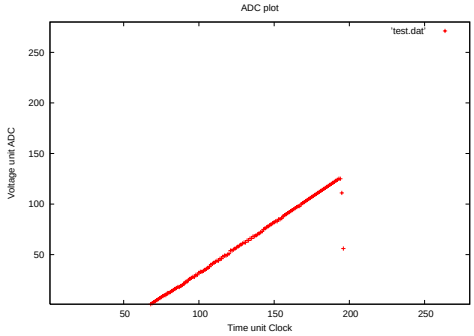


図 4.44: RAMP 関数のアナログ入力電圧

Status	value
Amplitude	2.0 V
Frequency	100 kHz

表 4.4: RAMP Status



以上の四つの関数について入力電圧と同じかどうかの評価を行う。見た目で形は似ていると分かるが、果してそれらは同じなのかということが疑問に残る。しかし、オシロスコープで見える入力信号波形と FPGA で再現した波形は同じようには見えないことがある。具体的には横に伸びるか縮むように見えることがある。これは ADC に入力されるクロックが原因である。クロックが遅い、すなわちサンプリングレートが小さくなると一つ ADC して、更に次の ADC を行うポイントまでの距離が長くなる。しかし今回は FPGA で再現を行う場合にそのことを考慮に入れないためその分縮んだように見える。図 4.49 では早いクロック、図 4.50 では遅いクロックの場合でのデータ取得後の表示例を表している。

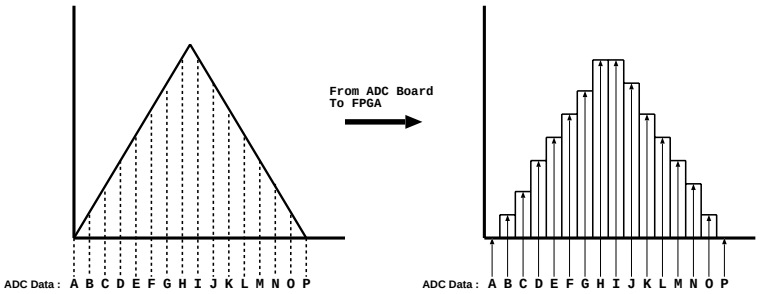


図 4.49: 早いクロックの場合でのデータ表示例

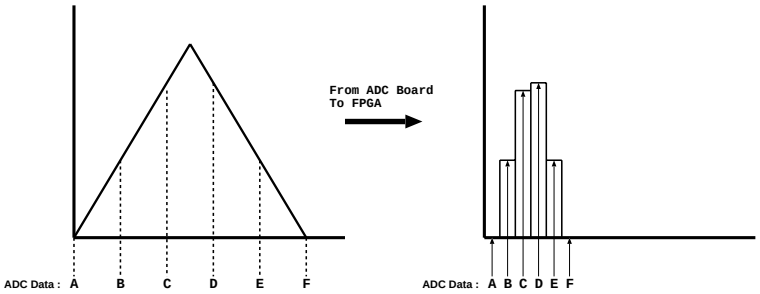


図 4.50: 遅いクロックの場合でのデータ表示例

そこでクロックが変化しているとしても変化しにくい場所について考察を行う。今回いずれの関数でも各々同じ条件で ADC を行い、FPGA で処理を行い、画像へと変換されたので、ピーク値とピーク位置は同じであるはずである。そこでこれらを調べて再現性が取れているかどうかを調べる。

図 4.51 から図 4.53 はどの場所をピーク値とピーク位置に定義としたかを簡単に描いている。SIN 関数のみピーク位置とピーク値が二つ得られたので二つ結果を載せている。

SIN

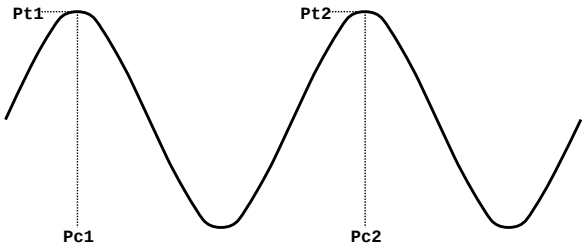


図 4.51: SIN 関数のピーク位置、ピーク値

	Pc1	Pt1	Pc2	Pt2
1	100	135	220	135
2	101	135	220	134
3	100	137	220	135
4	101	135	220	136

表 4.5: SIN 関数ピーク位置、ピーク値

TRIA

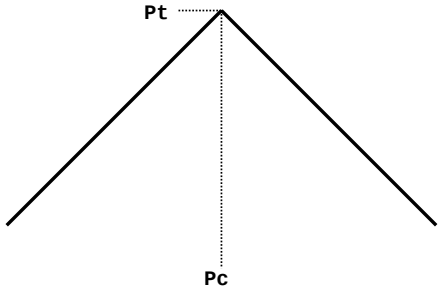


図 4.52: TRIA 関数のピーク位置、ピーク値

	Pc	Pt
1	133	126
2	133	126
3	132	126
4	133	126

表 4.6: TRIA 関数ピーク位置、ピーク値

RAMP

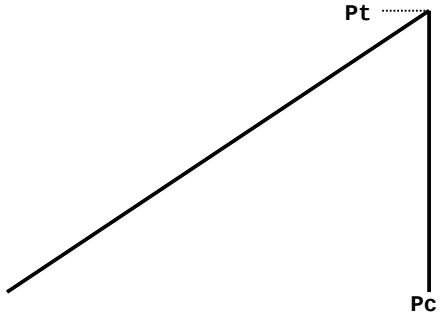


図 4.53: RAMP 関数のピーク位置、ピーク値

	Pc	Pt
1	194	125
2	194	126
3	194	124
4	194	125

表 4.7: RAMP 関数ピーク位置、ピーク値

入力波形の周期は SIN 関数が 200 kHz、TRIA と RAMP 関数が 100 kHz に対し、サンプリングレートは 24 MHz は十分に大きいため、チャンネルの誤差は ADC ボードでのサンプリングするクロックタイミングのずれのみであると言える。再現した図は ADC サンプリングクロックを元にプロットしているため、少なくとも 1 ch の誤差が生じる。しかし今回はいずれの関数でも差は 1 ch 以内である。この結果から常に同じ波形を取得することができることを表している。つまり Function Generator が常に同じ関数を出力して

いるということからADCボードは常に同じデジタル出力を行っているので、FPGAで同じような結果が出たということは波形を保存、再現することが成功したと言える。

4.6.2 Thresholdによる変化

入力された信号を再現することは出来たので次にRMAPでシステムのステータスを変化させたらどうなるのかということを考察する。RMAPによるステータス変化を行うことが出来ればリモートでの実験条件、測定環境を変化させることができる。これは人が簡単に行くことができない宇宙では必須の性能である。

本研究ではステータス変化を行い、かつそれが結果に表れるものはThreshold電圧ぐらいなのでこの値を変化させて結果を吟味する。入力電圧のステータスはFunctionがTRIA、Amplitudeが+3.8 V、Frequencyが80 kHzである。TRIAにしたのはこうすることでThresholdの変化が分かりやすいからである。

Thresholdは0x01から16ずつ増やし、0xFeまで変化させる。つまり0Vから+3.3 Vまで0.20625 Vずつ増やしていく。図4.54のようにThresholdを上げていくとTRI_out信号が立つ場所がどんどん信号上部になるため、取得した波形は全体的に左寄りになっていくと予想される。

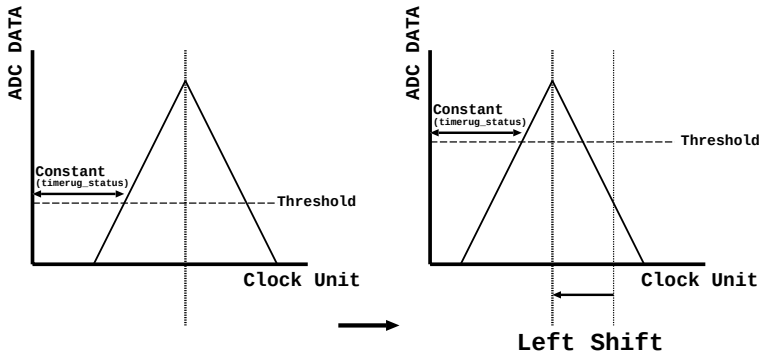
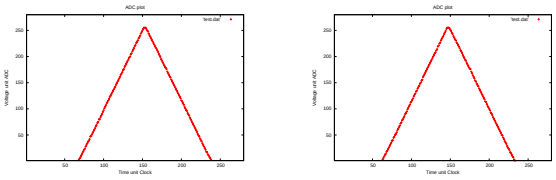


図 4.54: Threshold変更による取得した波形の変化予想図

以下が取得した波形である。図番号はThreshold値を示している。なお、0x00と0xffはそれぞれ下限、上限のため省いている。その代わりに0x01と0xfeの場合でのThresholdを載せている。


ADC plot
test.dat
Voltage unit ADC
Time unit Clock

ADC plot
test.dat
Voltage unit ADC
Time unit Clock

図 4.55: 0x01:01

FPGA 内での Threshold 値を 10 進法に直したもので、図 4.73 が実際の電圧に直したものである。縦軸が TRIA のピーク位置である。

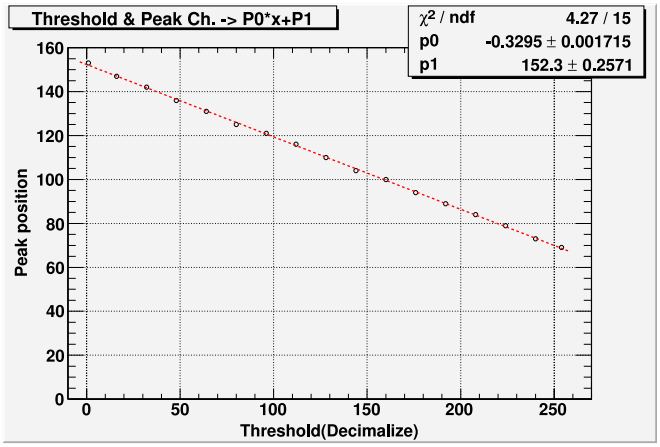


図 4.72: Threshold 値の変化に伴う TRIA 関数ピーク位置の移動

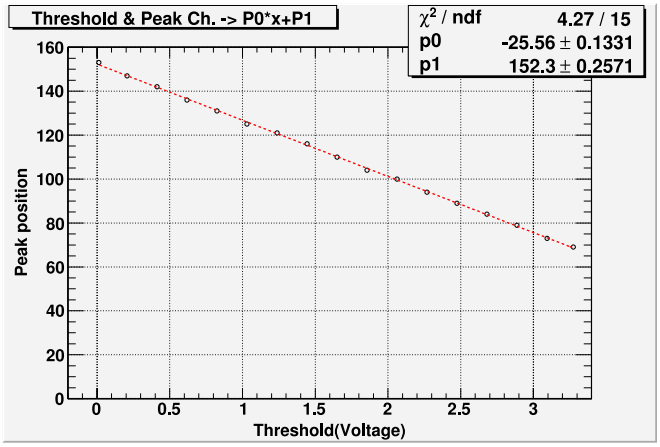


図 4.73: Threshold Voltage の変化に伴う TRIA 関数ピーク位置の移動

図から両者の関係はリニアであることが分かる。

以上より波形を ADS831 から出力された通りに保存していると言え、更に RMAP でのステータス変更もうまく機能させることが出来た。ADS831 には今回 Function Generator を用いているが例えば検出器などからの信号を入力すればライトカーブを描くことも可能であると言える。

第5章　まとめと今後

本研究ではADCボードを自作し、ADCされた信号をSpaceWire FPGA I/Oボード上にあるVHDLで回路設計を行ったFPGAで処理を行い、さらにSpaceWireを用いてSpaceCube、そして最終的にホストPCのLinuxPCへと転送を行った。

SpaceWireはケーブルを自作したが簡単にSpaceCubeとSpaceWire Digital I/Oボードと通信を行うことができたことより、非常に柔軟に接続できることが分かった。また波形を再現できたことより非常にデータが安定して通信できているともいえる。

信号処理を行う上で、ADCボードを自作したが、OP Ampが壊れていたもののアナログ回路の基礎やAD変換の仕組みを学ぶ上では非常に有意義であるといえる。FPGAに関してはVHDLの基礎、信号の処理、メモリマップ等、様々なデジタル処理の基礎を習得することができた。SpaceWireを用いた通信もメモリレジスタにアクセスするかのような手軽さであり、非常に汎用性に富むということが分かった。波形に乱れがないことからもOscillo Systemは波形保存モジュールとして性能がよいものであると言える。

しかしまだSpaceWireの基礎を学んだにすぎず、時間の取得、データ損失、データレイテンシの考慮などまだまだ開発すべき点は多い。今後はこれらのことを含めて考えれるようにし、Oscillo Systemなら改良できる点が多いのでそれらを再考する、もしくは新たに別のSystemを開発し、FPGA、VHDL、そしてSpaceWireについてより深い知識を得たい。

謝辞

本研究を進めるにあたってまずM2の田中さんに心から感謝します。田中さんにはFPGA、VHDL、ADCボードの製作、SpaceCubeの基礎を教えて頂きました。またトラブルがあったときも忙しい合間に時間を作ってもらって一緒に考えていただいたりと感謝し尽くせません。ありがとうございます。大杉先生には研究室や、これからの人生での在り方を教えていただき、また検出器での原理なども教えていただき、ありがとうございます。深沢先生には環境の整備、的確なアドバイス、遠い部屋にいるのにも係わらず最後までしっかりとした御指導に感謝しています。

M1の西野さん、D1の安田さんにもDSSDの基礎を教えていただいてとても助かりました。感謝しています。また事務の上原さんには手続き等、分かりやすい説明もしていただきました。ありがとうございます。また4年の池尻君には同じ実験室で協力や、励まし合いながら研究を行うことができました。ありがとうございました。他の研究室のみなさんも、みなさんのおかげで充実した研究を行えたと思います。みなさんに感謝します。

関連図書

[1] 高橋忠幸 笠羽康正 高島健 吉光徹雄 山田隆弘 能町正治 「科学衛星データ処理系の将来展望」

[2] 田中琢也「宇宙 X 線観測用放射線検出器多チャンネル読み出しシステムの開発」、卒業論文 広島大学、2006

[3] Hirokazu Odaka Takayuki Yuasa「SpaceWire のつなげかた TRON 版 SpaceCube 編」

[4] サンハヤト社 「『ポジ感光基板』を使ったオリジナル基板製作の手順」

[5] 長谷川裕恭 「VHDL によるハードウェア設計入門」 CQ 出版

[6] ECSS, Space engineering ECSS-E-50-12A, 2003

[7] 次期 X 線天文衛星計画ワーキンググループ 「NeXT 計画提案書」,2005

[8] 澤本直之 「シリコンストリップ検出器の X 線応答と多チャンネル読み出し回路のパラメータ調整」, 広島大学, 卒業論文,2004

[9] 西野 翔 「NeXT 衛星搭載硬 X 線撮像用両面シリコンストリップ検出器の性能評価」, 広島大学, 卒業論文,2007

[10] トランジスタ技術 2001 年 5 月号 CQ 出版社

[11] トランジスタ技術 2006 年 12 月号 CQ 出版社

[12] Xilinx 社,ISE 9.1 i アドバンスチュートリアル

[13] Xilinx 社,Spartan-3 ジェネレーション FPGA ユーザーガイド (UG331 v1.2),2007

[14] Xilinx 社,Platform Flash インシステム プログラマブル コンフィギュレーション PROM(DS123 v2.11),2007

[15] Personal Media Corporation,Teacube/VR5701 評価キット取扱説明書 1.A0.01, 2004

[16] シマフジ電機株式会社,SPW Digital I/O ボード (双方向拡張バス、uart 機能付き) 取り扱い説明書 ver 0.3, 2007

[17] Texas Instruments 社,OPA691 Data sheet,2006

[18] Texas Instruments 社,ADS831 Data sheet,2001

[19] Toshiba,TC74LCX541F Data sheet,1997

[20] Toshiba,TA48M033F Data sheet,2007