2024 年度 修士論文

シリコン検出器読出高速化に向けたプロセッサ内蔵型 FPGA を用いた回路設計

先進理工系科学研究科学 先進理工系科学専攻
物理学プログラム

博士課程前期

M224873

添田 拓

指導教員　山口 頼人 准教授
主査　山口 頼人 准教授
副査　高橋 弘充 准教授
副査　檜垣 浩之 准教授

2024 年 1 月 31 日

概要

近年、高エネルギー加速器の高輝度化が進んでいる。加速器の高輝度化によって、データ量の増加という大きな利点が期待できるが、急増したデータをオフライン処理のために全てストレージに保存するのは現実的でない。そのため、検出器読出およびデータ伝送の高速化・大容量化は必要不可欠である。このような高速かつ大容量な検出器読出・データ伝送技術は物理実験に限らず、産業用非破壊検査などに応用できる技術である。
本研究ではシリコン検出器を用いた産業用非破壊検査システムの高速・大容量データ伝送化に向けて SoC-FPGA (System-on-Chip FPGA) を用いた基本システムを設計し、制御・読出・データ伝送の簡略化を行なった。柔軟なデジタル回路設計が可能な FPGA に加え、CPU・メモリコントローラ・多彩なインターフェースを 1 つのチップ内に統合した SoC-FPGA によりハードウェアアクセラレーションに必要なマルチタスクを効率的に行うことが可能になる。
SoC-FPGA を用いた読出システムには、4 つの開発要素がある。

1. センサや ADC などの外部機器を制御するモジュールの設計
2. 一時的にデータを貯蔵するバッファの設計
3. 貯蔵したデータをメモリに一括転送するスキームの設計
4. メモリに転送されたデータを外部 PC に転送するネットワークプログラムの設計

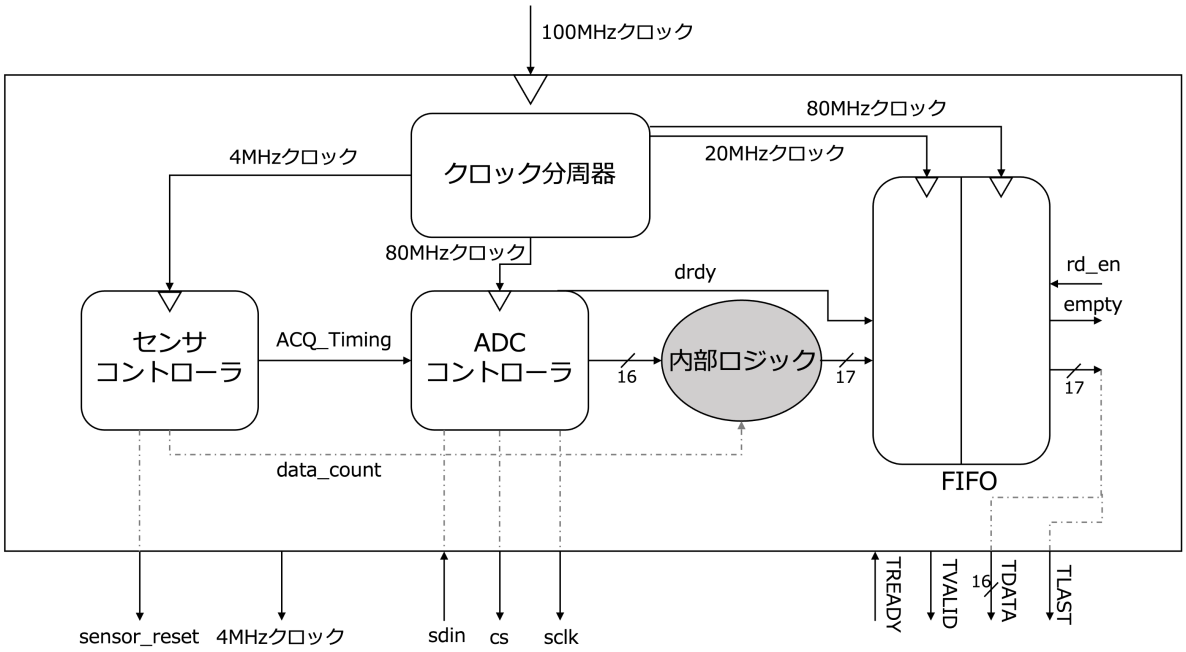


図 1　設計した FPGA ロジックのブロック図

回路設計は、ハードウェア記述言語である Verilog, System Verilog でプログラム化される。開発環境に Vivado を用い、シミュレーションを通して動作確認を行なった。回路設計はハードウェア記述言語でプログラムし、シミュレーションを通して動作確認を行なった。直列 256 チャンネルのセンサ出力を 1 サイクルとし、連続的に 1 サイクルのデータを繋げて検査物を X 線画像化する必要があるため、シミュレーション結果を元にセンサを各チャンネルの ADC 変換過程レベルからタイミング設計を行なった。外部 PC へのデータ伝送には、A/D 変換などを行う PL(Programmable Logic) からデータ処理などを行う PS(Processing System) にデータを転送する必要があり、CPU を介さずに効率的な転送が可能な DMA(Direct Memory Access) 方式を採用した。DMA 転送の前にはセットアップが必要なので、センサからデータが出力されるたびに DMA 転送をしては、オーバーヘッドの影響でデータの高速読出は成し遂げられない。まとまった量のデータを DMA 転送するために、バッファが必要不可欠である。DMA 転送の所要時間をテスト回路を用いて実測し、センサ・ADC の出力にかかる時間を考慮してシミュレーションを行なって、適切なバッファ容量やバッファの書込み・読出しのトリガーを設計した。メモリに転送されたデータはネットワークを通して外部 PC に送信されるため、ソケット通信のプログラムを作成し、基本的なシステム設計を完成した。

目次

第 1 章

序論

1.1 原子核衝突実験

素粒子物理学は、宇宙の最も基本的な構成要素とそれらの相互作用の理解を目指す学問である。実験分野では加速器を用いて陽子や電子、金や鉛などの重イオンを光速に近い速度まで加速し、高エネルギーでの原子核衝突実験を通して素粒子の性質や新粒子、希少な物理現象を探索している。超高エネルギーに加速された原子核同士が衝突することで、瞬間的に運動エネルギーや束縛エネルギーなどの巨大なエネルギーが解放される。アインシュタインの相対性理論 $E = mc^2$ より、このエネルギーは質量に変換される可能性がある。図 1.1 は、特に重イオン衝突の様子を示したものである。このように、重イオン衝突では解放されたエネルギーによって様々な種類の素粒子が多数生成される。欧州原子核研究機構 (CERN) における LHC(Large Hadron Collider) はスイス・フランス国境にまたがる周長 27km の世界最高エネルギーを誇る加速器であり、様々な実験グループが参加している。中でも ALICE 実験は重イオン衝突実験に焦点を当てており、ビッグバン直後の初期宇宙に存在したと考えられているクォーク・グルーオン・プラズマ (QGP) 状態の理解という重要な目標を掲げている。QGP は、クォークとグルーオンが個々の粒子として自由に動き回る高温・高密度を指し、現在の宇宙において通常の物質とは異なる特異な相である。重イオン衝突実験では、超高エネルギーまで加速された鉛などの原子核を衝突させ、この QGP 状態を再現して宇宙初期や物質の基本的な性質や相互作用の理解を目指している。

加速器を用いた高エネルギー重イオン衝突実験は、高度な技術と国際的な協力を必要とする。これらの実験の成功は、精密な検出器の設計と構築、膨大なデータを収集する読出システムの設計、データ解析フレームワークの開発、そして理論物理学者と実験物理学者、これら全ての緊密な協力に依存している。

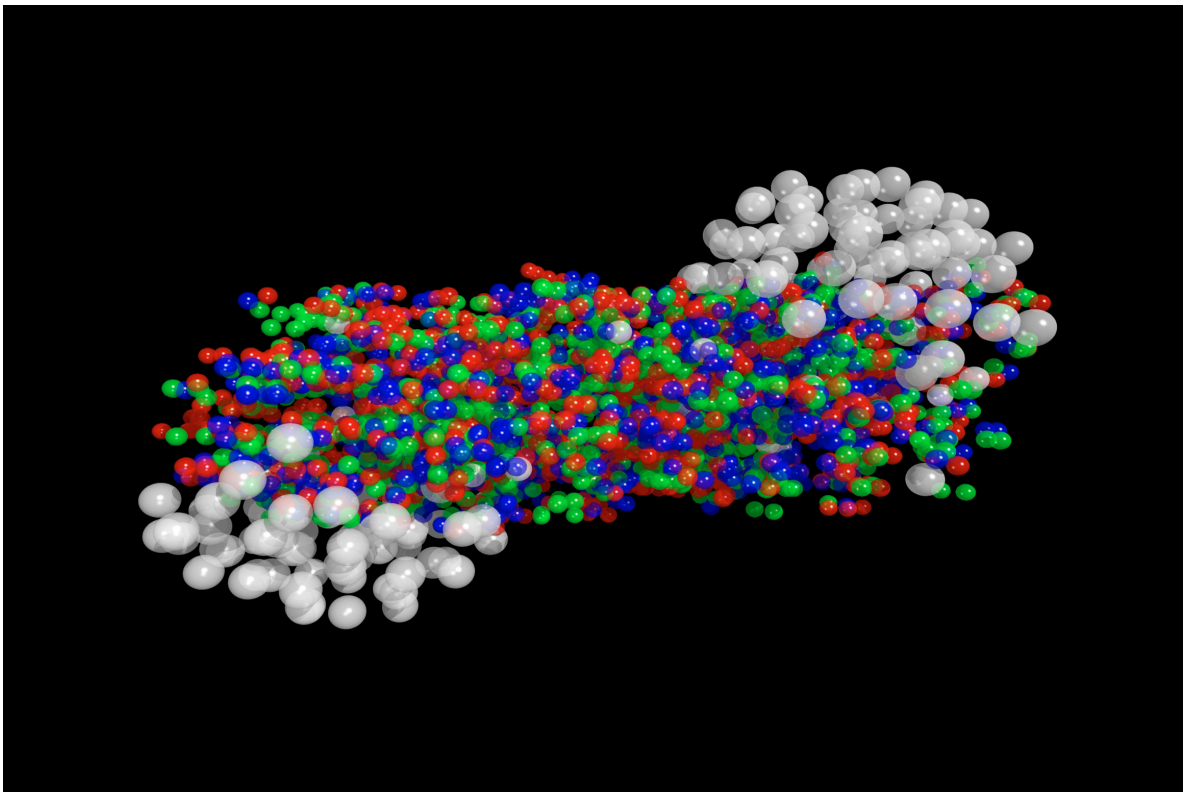


図 1.1　原子核衝突のようす [7]

1.2　加速器の高輝度化

近年、粒子加速器技術においては、高エネルギー衝突のみならず、高輝度化への関心が高まっている。輝度 (Luminosity) とは、単位時間あたりの粒子衝突数を示す指標であり、式 1.1 のように表される。ここで、$\frac{dN}{dt}$ は単位時間あたりに検出された粒子衝突事象の数、σ は反応断面積 (cross section) といい、単位面積あたりでの粒子衝突が発生する確率である。

$$L = \frac{1}{\sigma} \times \frac{dN}{dt} \qquad [cm^{-2}s^{-1}] \tag{1.1}$$

図 1.2 に示すように、高輝度化とはビーム強度やビーム密度の向上により、単位時間・単位面積あたりの衝突イベントが増加することを意味している。ここで、一つ一つの衝突は、図 1.1 に相当している。このアプローチは、特にまれな物理現象や微弱な信号を探索する際に非常に重要である。高輝度の加速器を用いた原子核衝突実験は、より多くのデータを生成し、統計的有意性の高い結果を得る可能性を高める。

このように、加速器の高輝度化は素粒子物理学実験に対して重要な役割を果たすことが期待されている反面、検出器の時間・エネルギー・位置分解能の向上やデータ読出システムの高度化という課題がある。

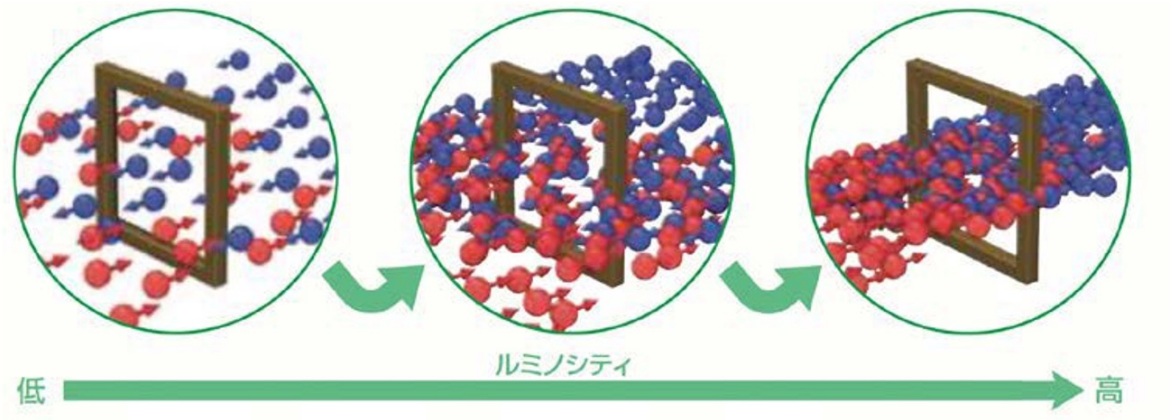


図 1.2　加速器の高輝度化イメージ [1]

1.3　シリコン検出器

加速器の高輝度化は、検出器技術に対して新たな要求をもたらしている。短い時間間隔で多数の衝突が発生するため、高い水準の分解能が要求されている。個々の衝突事象を正確に区別するための時間分解能、検出される粒子の種類や衝突の特徴を詳細に観測するためのエネルギー分解能、散乱粒子の軌道や粒子衝突点の正確な特定のための位置分解能が挙げられる。粒子衝突実験では荷電粒子の検出器として、ガス検出器・シンチレーション検出器・半導体検出器がよく用いられる。

図 1.3 は、一般的な PN 接合型半導体検出器の例である。正孔をキャリアとして持つ P 型半導体と、自由電子をキャリアとして持つ N 型半導体を接合して用いている。それらの境界付近では、正孔と電子が再結合することで電場を生じる空乏層が形成される。放射線や荷電粒子が PN 接合型半導体検出器に入射すると、そのエネルギーに応じた電子-正孔対が生成され、内部電場によってそれらが読出電極に到達することで検出が行われる。

半導体検出器は、電子-正孔対の生成に必要なエネルギーが他の 2 つに比べて低いため、高いエネルギー分解能を持つ。また、近年の技術の進歩により検出素子の小型化・薄型化が進んでいる。電子-正孔対が読出電極に到達することで検出が行われるが、読出電極までの距離が短くなったことにより高い時間分解能を、より素子密度の高い検出器を構成できるため高い位置分解能を実現可能である。それらの要求を高水準で満たす半導体検出器のうち、比較的安価に入手が可能なシリコン検出器は、高輝度粒子衝突実験において有力な検出器として考えられている。

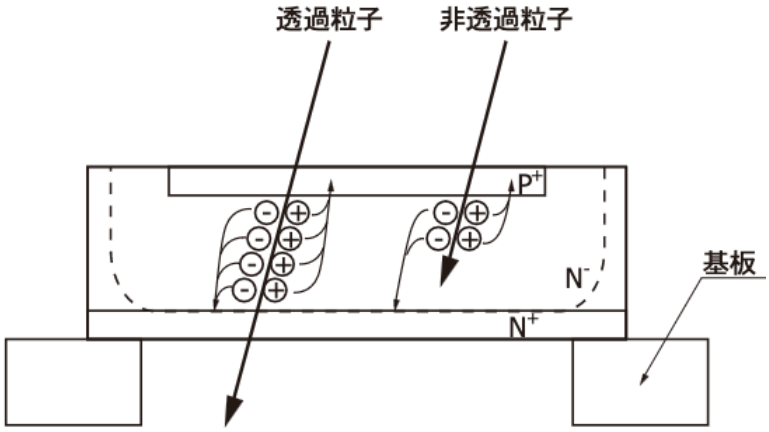


図 1.3　PN 接合型半導体検出器 [11]

1.4　データ読出技術の課題

加速器の高輝度化に伴い、データ読出技術における課題は顕著になっている。図 1.4 は、LHC を用いた高エネルギー重イオン衝突実験である ALICE 実験で現在用いられている検出器群である。これらの検出器の生データは、連続読出により 3.5 - 4TB/s にまでのぼり、現時点でもハードウェアアクセラレーションが必要である。加えて、

- 加速器の高輝度化による衝突事象の増加
- シリコンセンサの小型化による検出器の高密度化 (1.3 節)

によって、極めて短時間内に多数の原子核衝突事象と生データの増加によって、単位時間あたりのデータサイズが急増することが予想される。ゆえに、ハードウェアアクセラレーションを含めた、高いデータレートのサポートとデータの信頼性の担保、そして大容量のデータ転送が実現可能なより高度な読出技術が必要不可欠である。それらの課題に対して、以下のような効果的な手法が挙げられる。

データのリアルタイム処理　システム中に蓄積されるデータ量の削減がにより、ストレージを有効に活用できる。また、システム全体の効率向上により、データ転送の遅延やデータ処理によるボトルネックの影響を減少させることができる。

フィルタリング　大量のデータの中に含まれるノイズの効果を減少させるとともに、処理するデータ量を削減することで後続のプロセスの効率化を図ることが可能。

データの圧縮　データのパターンや冗長性をより効率的な形で表現することで、同じ量のデータをより少ない形式で扱うことが可能になる。ネットワーク経由での転送速度

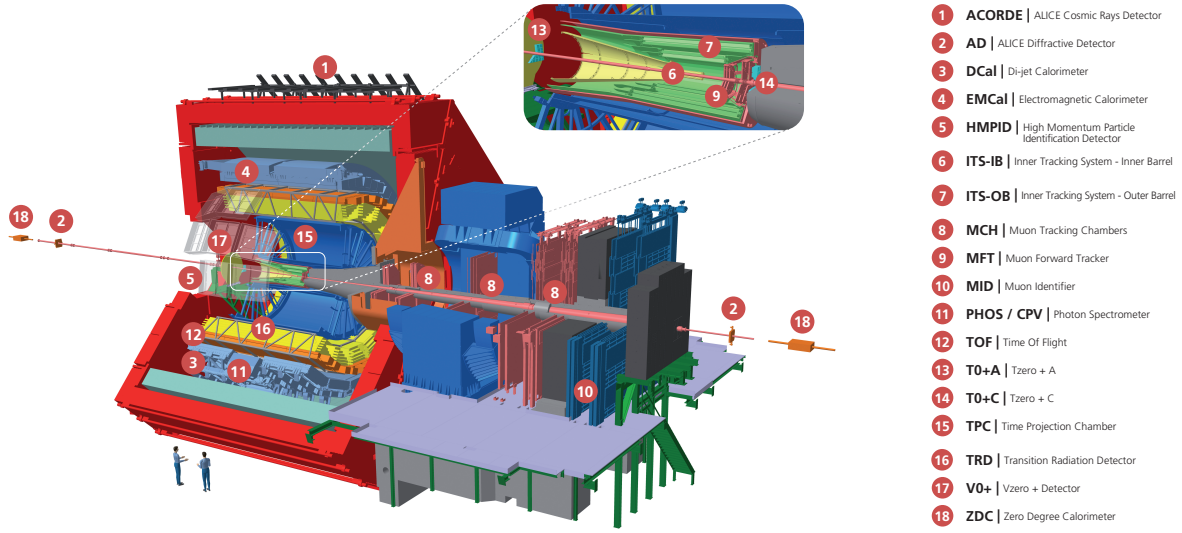


図 1.4　ALICE 実験 Run 3 における検出器群 [2]

向上や、通信の帯域幅の削減効果によりネットワークの負荷を軽減し、他タスクの性能向上が期待できる。大量のデータのアーカイブにおけるプロセスを簡素化できる効果もある。

GPU を用いたハードウェアアクセラレーション　本来グラフィックス処理に特化して設計されたが、強力な並列列処理特性が注目されている。単純な計算の繰り返しや、ディープラーニングを用いた大量の行列計算などにより、複雑なデータ処理に対する強力な回答が期待できる。

大容量データ転送　後続の解析プロセスにネットワークを通じてデータを転送する際に、単位時間あたりに転送可能なデータサイズをより大きくする。これにより、データの損失や読出回路によるボトルネックの影響を削減し、後続のハードウェアの性能を十分に利用可能になる。

1.5　FPGA についての基礎知識

FPGA (Field Programmable Gatte Array) とは、ユーザーが特定の用途に合わせてプログラムが可能な半導体デバイスである。FPGA は、多数のロジックブロックで構成される極めて設計自由度の高い集積回路である。図 1.5 にロジックブロックの構造を示した。ロジックブロックとは、特定の入力に対して事前に定義された出力を提供するためのテーブルであるルックアップテーブル (LUT)、ビット情報を一時的に保持するフリップフロップ (FF) やレジスタ、AND・OR・NOT の論理ゲートから構成される。ユーザーはロジックブロックの接続や構成、動作を自由に変更し、それらを並列に動作させること

が可能である。したがって、FPGA はハードウェアレベルでのカスタマイズが可能であり、特定のアプリケーションやタスクなどに対して高度に最適化された処理を高速に並列して行うことが可能である。その柔軟性と高速性、並列処理への強みから、検出器読出の現場では FPGA が一般的に用いられている。

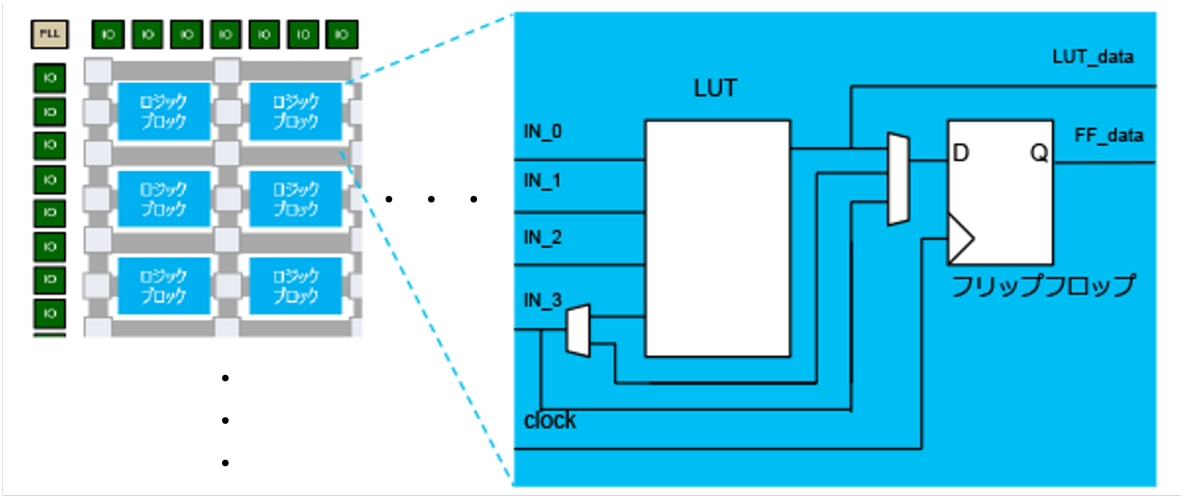


図 1.5　FPGA のロジックブロック構成と基本的なロジックブロックの構造 [6]

1.6　研究動機・開発目標

本研究のもおく的は高輝度加速器実験でのシリコン検出器読出高速化に向けた基礎回路設計である。既存の産業用非破壊検査用シリコン検出器システムを実例として読出高速化の核となる基本回路設計を行った。物理実験と非破壊検査では細部の仕様が異なるものの、大量かつ高速でのデータ処理・読出に必要な基本回路は同等である。この要求に対応するために、SoC-FPGA(System on Chip FPGA) を用いた回路設計に取り組んだ。System on Chip とは、プロセッサやメモリ、各種周辺デバイスのコントローラなどを 1 つのチップに集積していることを意味する。詳細は 2.2.3 節、図 2.9 を参照してほしい。1.5 節で説明した FPGA に加え、プロセッサ機能が集積されていることから、FPGA の持つ柔軟性・高速性を活かしつつ一般的なコンピューティングタスクが実現可能である。これは、データ処理アルゴリズムと迅速なデータ読出というタスクに特化した回路が設計できることを意味している。また、FPGA を用いた回路と CPU(PC) を PCIe(Peripheral Component Interconnecr Ecpress) 接続して用いる一般的な読出回路と比べ、FPGA ブロックと CPU ブロックとでダイレクトに通信が可能なことから、物理的・アーキテクチャ的にコンパクトな、密で効率的な読出システムが実現可能である。こ

のように、SoC-FPGA を用いた検出器読出はハードウェアアクセラレーションに向けた主要な要素として有効であり、将来的な検出器読出システムの基礎となるものである。

本研究は、産業用の X 線非破壊検査装置を扱う企業と協力して行なった。X 線非破壊検査は、製品の品質保証や構造的な欠陥の検出を目的に一般的に利用される手法であり、この分野においても高速で正確なデータ処理は単位時間あたりに検査可能な製品数に直結するが非常に重要な要素である。検査物の中身を検査するために、物質の透過能力に優れる X 線を使用する都合、シリコン検出器を採用している。1.3 節で述べた通り、シリコン検出器は高い解像度を提供する反面、同時に読出速度の高速化や高度なデータ処理の実装が課題になっている。

したがって、本研究の焦点は SoC-FPGA を用いてシリコン検出器の読出回路を設計し、高度データ処理の実装に繋げることにある。このアプローチにより、高輝度粒子衝突実験と産業用の X 線非破壊検査の両方におけるデータ読出およびデータ処理の課題に対処し、より高度な読出システムを実現することを目指している。

第 2 章

回路設計

1.6 節で述べた通り、本研究では産業用 X 線非破壊検査で使用されているライン型シリコンセンサを、SoC-FPGA を用いて読み出すための回路を設計している。以下の開発要素に沿ってモジュール設計を行ない、図 2.1 のような全体の回路を設計した。表 2.1 は全体回路の信号説明である。詳細な説明は 2.3 で述べてある。

1. センサや ADC などの外部機器を制御するモジュールの設計
2. 一時的にデータを貯蔵するバッファの設計
3. 貯蔵したデータをメモリに一括転送するスキームの設計
4. メモリに転送されたデータを外部 PC に転送するネットワークプログラムの設計

図 2.1 は、センサコントローラ、ADC コントローラ、クロック分周器、FIFO(バッファ)の 4 つのブロックで構成されており、センサ・ADC との通信が可能である。FIFO に書き込まれたデータは、2.3.5 節で紹介する DMA 転送を行うことでメモリへのデータ内部転送が可能になっている。メモリに内部転送されたデータは、2.2.4 節で述べるネットワークを通じて外部 PC へと転送される。

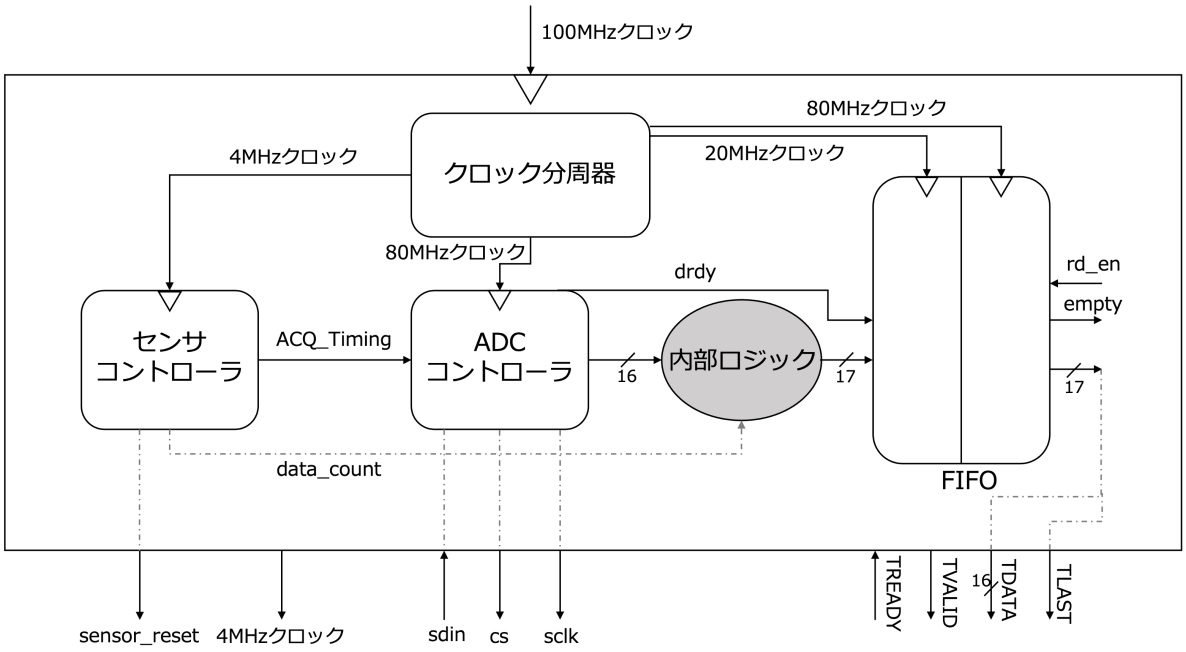


図 2.1　設計した全体回路

信号名	I/O	To/From	説明
sensor_reset	O	センサ	センサ駆動パルス
4MHz クロック	O	センサ	センサ同期クロック
sdin	I	ADC	シリアルデータ
cs	O	ADC	ADC 変換プロセス開始パルス
sclk	O	ADC	シリアルデータ送信同期用クロック
TREADY	I	DMA コア	DMA コア転送準備完了フラグ
TVALID	O	DMA コア	FIFO 読出データ有効フラグ
TDATA	O	DMA コア	FIFO 読出データ
TLAST	O	DMA コア	FIFO 読出最終データフラグ

表 2.1　全体回路の入出力信号

2.1 開発環境

以下では、本研究で使用した機器および開発ソフトウェアについて述べる。

2.1.1 使用機器

センサ 本研究では、X 線ライン型センサ (図 2.3) を使用している。一般的に、X 線センサは物体の内部を視覚化するために用いられている。図 2.2 のように、検査物を透過した X 線の量をセンサで測定し、必要に応じてコンピュータ処理を施すことで、透過量に応じた白黒の濃度で物体の形状や内部構造を X 線画像化する。ここで述べている X 線画像とは、検査物を連続的な薄いスライスに切り分けて撮影し、それぞれのスライスを連続的な一枚の画像に再構成したものである。本研究で用いた X 線ライン型センサは、128 個のシリコンフォトダイオード素子やその他の制御回路からなるフォトダイオードアレイ (図 2.4(b)) が 2 つで構成される直列 256 チャンネルセンサである。タイミング発生回路が内蔵されているので指定されている信号を入力するとセンサが動作を開始する。このセンサは、データとして各素子に蓄積された電荷量に応じたアナログ信号、Video 信号を出力する。この信号を ADC を通して読出回路に渡すことで、前述した X 線画像を取得できる。

ADC Digilent 社の Pmod (Peripheral Module) シリーズの ADC モジュールである、Pmod AD1(図 2.5) を使用した。Pmod AD1 は 12bit の解像度と最大 1MHz のサンプリングレートを持ち、最大で 2 つの信号線を同時に変換が可能である。後述する開発ボード、Eclypse-Z7 の Pmod ポートに挿して使用し、SPI (Serial Peripheral Interface) ベースの通信インターフェースを使用して制御を行う。

開発ボード Digilent 社の開発ボード、Eclypse-Z7(図 2.6) を使用した。このボードは、特にアナログ信号の処理や制御しシステムの開発に焦点を当てて設計されている。本研究では X 線非破壊検査に向けたデータ読出を目標に設定したことから、Eclypse-Z7 を採用した。主な特徴として、1.6 で述べた SoC アーキテクチャ、Pmod インターフェース、イーサネットポートが挙げられる。SoC-FPGA は、Xilinx 社の Zynq-7000 シリーズを搭載している。図 2.9 に示すように、Zynq-7000 は ARM 社の CPU、Cpretex-A9 プロセッサが統合されており、FPGA のプログラム可能なロジックブロックと合わせることで高速なプロセシングに代表されるカスタムハードウェアアクセラレーションや特定のタスクのためのロジック回路の実装が可能になっている。Eclypce-Z7 と Zynq-7000 の関係は、ちょうどマザーボードと CPU のようなものであり、Eclypce-Z7 ボードには DDR メモリや Pmod ポート、イーサネットポートが実装されている。それら周辺機器と Zynq-7000 は

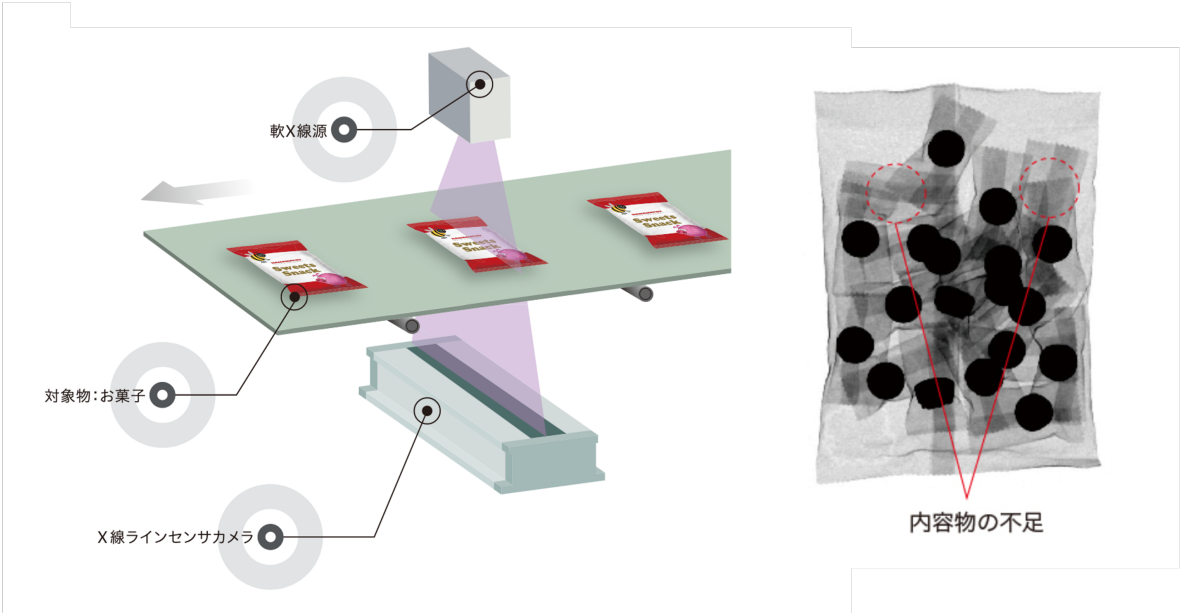


図 2.2　X 線非破壊検査例と X 線画像 [12]

バスを通して直接通信が可能である。それにより、センサのデータを Pmod AD1 を通して Zynq-7000 に入力し、内部で処理を施したのちにネットワークを通して外部 PC に伝送することが可能になる。

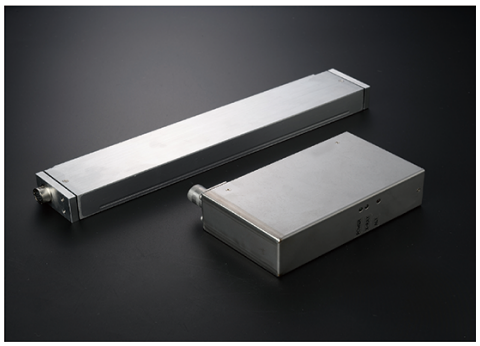
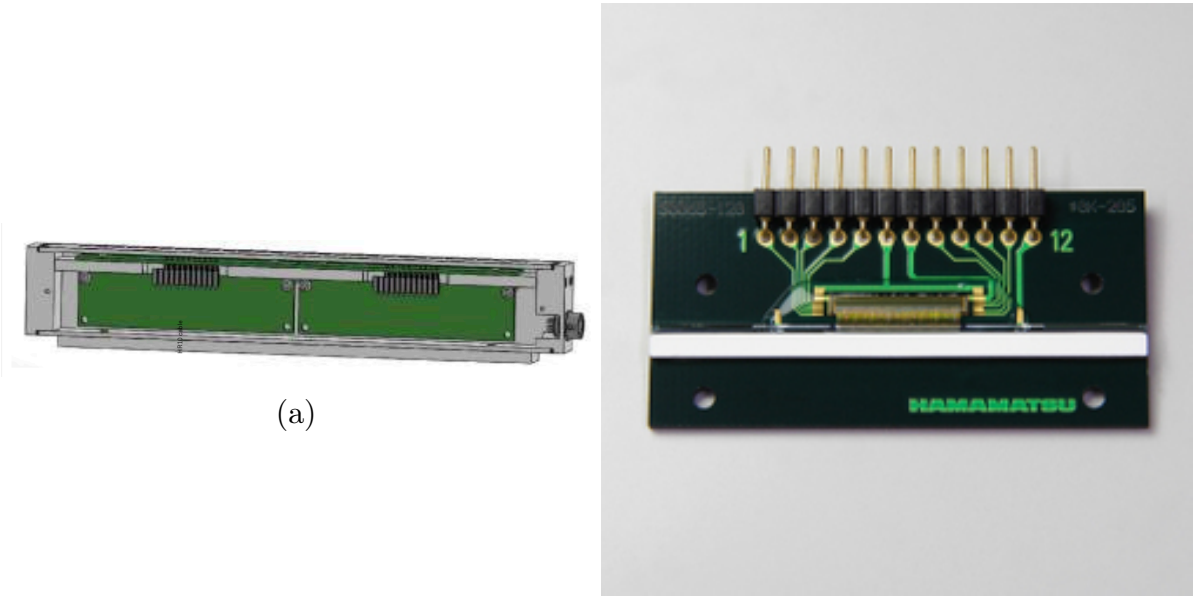


図 2.3　X 線ライン型センサ (写真左) [10]

2.1.2　開発ソフトウェア

Xilinx 社が提供している FPGA 開発環境である、Vivado を用いている。回路記述には、Verilog や VHDL などの HDL (Hardware Description Language) を用いる。通常のソフトウェアプログラミングと異なる点として、基準となる信号の立ち上がりや立ち下



(a)

(b)

図 2.4　センサ構成要素: (a) 概形 (b) 内部フォトダイオードアレイ

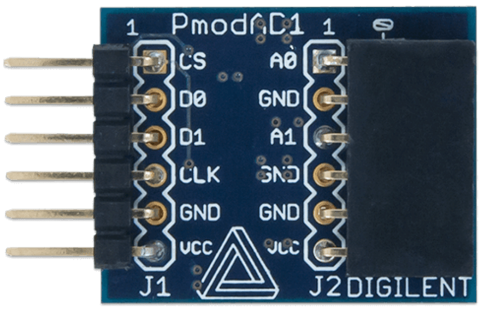


図 2.5　Pmod AD1 [5]

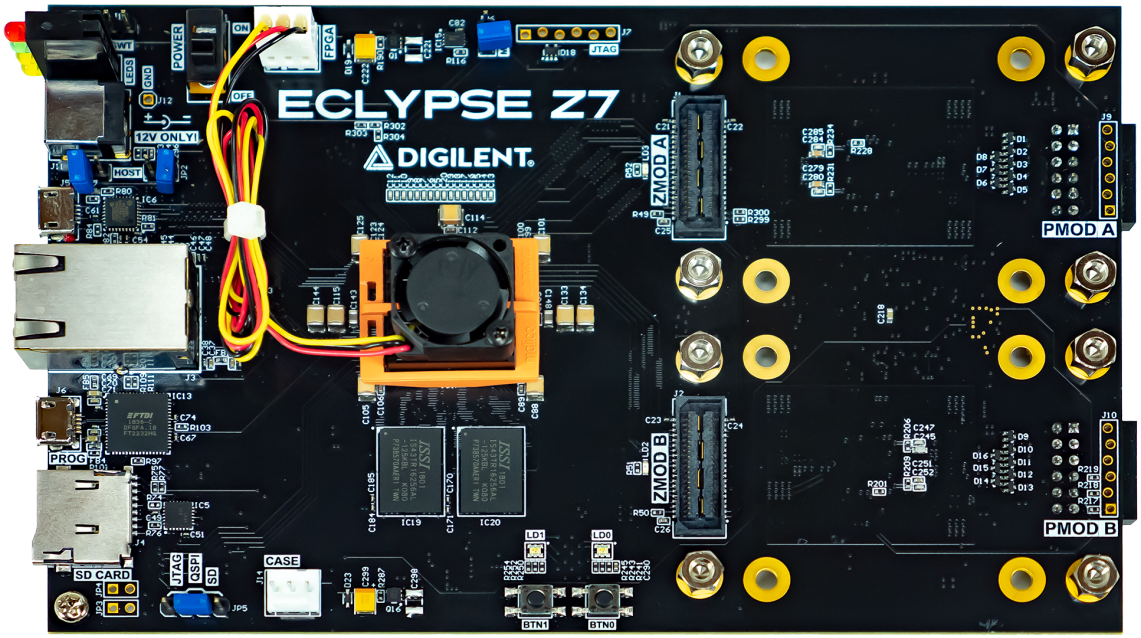


図 2.6　Eclypse-Z7 [4]

がりのタイミング、または状態や値でその他の信号の動作を記述する点が挙げられる。このようにして設計した動作をするモジュールを開発できる。一般的に、モジュールは一定の機能単位で作成する。作成したモジュールは、階層構造をとって組み合わせ、全体の回路設計を行う。また、Xilinx 社やその他のサードパーティベンダーがすでに設計・検証を終えた回路である IP コア (Intellectual Property) を提供している。パラメータや入出力ポートをカスタマイズし、他のモジュールとの信号の接続を行えば自分の回路に組み込むことが可能であり、自ら設計したモジュールを IP 化して再利用することも可能である。回路の設計が終了したら、Vivado の回路シミュレーション機能を用いて動作確認を行う。データやクロック・リセット信号などの入力信号をプログラム上で設計し、回路に入力して内部の信号やレジスタの動作やタイミング、状態などの確認が可能である。以上のようにして回路設計が完了したら、FPGA に読み込ませるバイナリファイルを生成する。この際、信号の入出力に用いる Pmod ポートなどのピンアサインを行い、Vivado によって FPGA リソースの最適化が行われる。

2.2　開発要素

2 章冒頭で述べた 4 つの開発要素について詳細に取り上げていく。

2.2.1 センサコントローラ

センサ駆動のための制御信号を生成したり、その他後続の回路で用いる信号を生成する回路である。入力信号は、同期用のクロック信号とセンサのスタートパルスとなるリセット信号である。それらの信号を受けてセンサは動作を開始し、合計 256 チャンネル分のデータを出力し続ける。

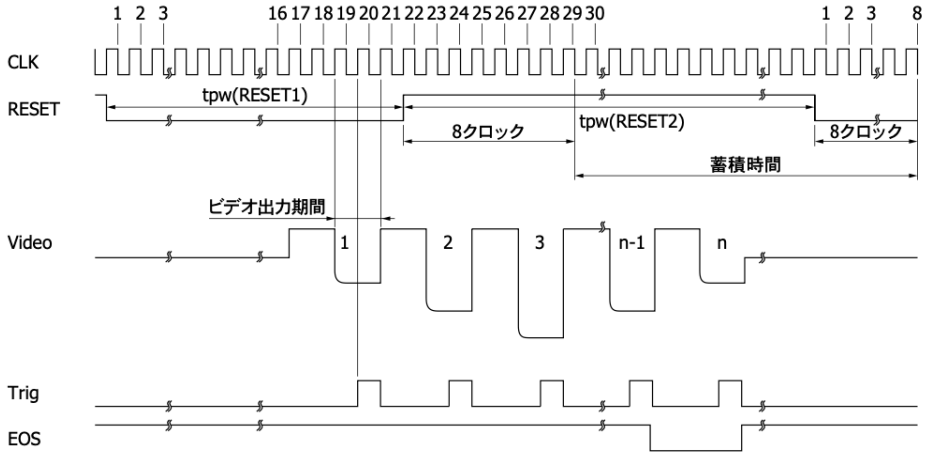


図 2.7　センサ動作タイミングチャート [10]

信号名	I/O	説明
CLK	I	同期用のクロック信号
RESET	I	センサの動作開始を指示する信号
Video	O	フォトダイオード素子の出力信号
Trig	-	推奨データ取込タイミング
EOS	-	センサ内部回路用信号

表 2.2　図 2.7 の信号

図 2.7 はセンサの動作を、表 2.2 はその信号の説明を示したものである。RESET 信号の立ち下がりタイミングから 18 クロック後経過すると、Video 信号が 4 クロック毎に 2 クロックの間出力される。本研究では 256 チャンネルのセンサを使用しているため、式 2.1 より、次に RESET 信号を立ち下げるまでには最低 1040 クロックの時間を確保する必要がある。

$$18 + 2 + 4 \times (256 - 1) = 1040 \tag{2.1}$$

2.2.2 ADC コントローラ

Pmod AD1 の解像度は 12bit だが、信号の先頭に 0000 の 4bit を加えて 16bit として扱っている。これは開発元の Digilent 社の定めている仕様であり、変更はできない。多くのデジタルシステムでは 2^n Byte 単位 (8, 16, 32, 64bit) のフォーマットを使用しているため、12bit よりも 16bit フォーマットでデータ処理をする方が多くのシステムとの互換性が高くなるためだと考えられる。

このコントローラは Pmod AD1 の動作のための制御信号を生成し、Pmod AD1 が出力するシリアルデータ (1bit のデータ) を 16bit にまとめ、後続の回路に出力する回路である。

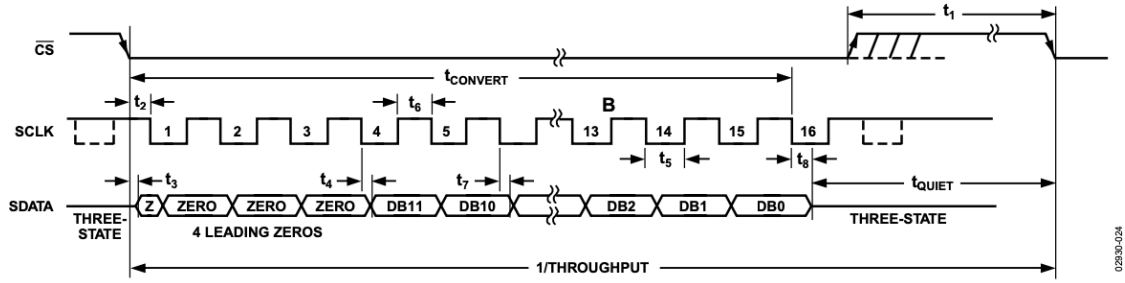


図 2.8　ADC 動作タイミングチャート [3]

信号名	I/O	説明
CS	I	データ変換プロセスの開始を指示する信号
SCLK	I	データ送信の同期用シリアルクロック
SDATA	O	シリアルデータ出力信号

表 2.3　図 2.8 の信号

図 2.8 はセンサの動作を、表 2.6 はその信号の説明を示したものである。CS 信号の立ち上がり ($\overline{CS}$ 信号の立ち下がり) を受けてアナログデータのサンプリングおよび A/D 変換プロセスが開始される。この間、1 つ前のサイクルにサンプリングされ A/D 変換されたデータ (SDATA) が SCLK 信号に同期して 1bit ずつ出力される。以上より、ADC コントローラは CS・SCLK を仕様通りに出力し、Pmod AD1 が出力する SDATA を受け取りながら 16bit データにまとめる必要がある。

2.2.3　バッファ・データ内部転送スキーム

まず初めに、本研究で用いている SoC-FPGA チップである Zynq-7000 の構造 (図 2.9) を確認する必要がある。上半分と下半分で背景色が異なって示されているように、

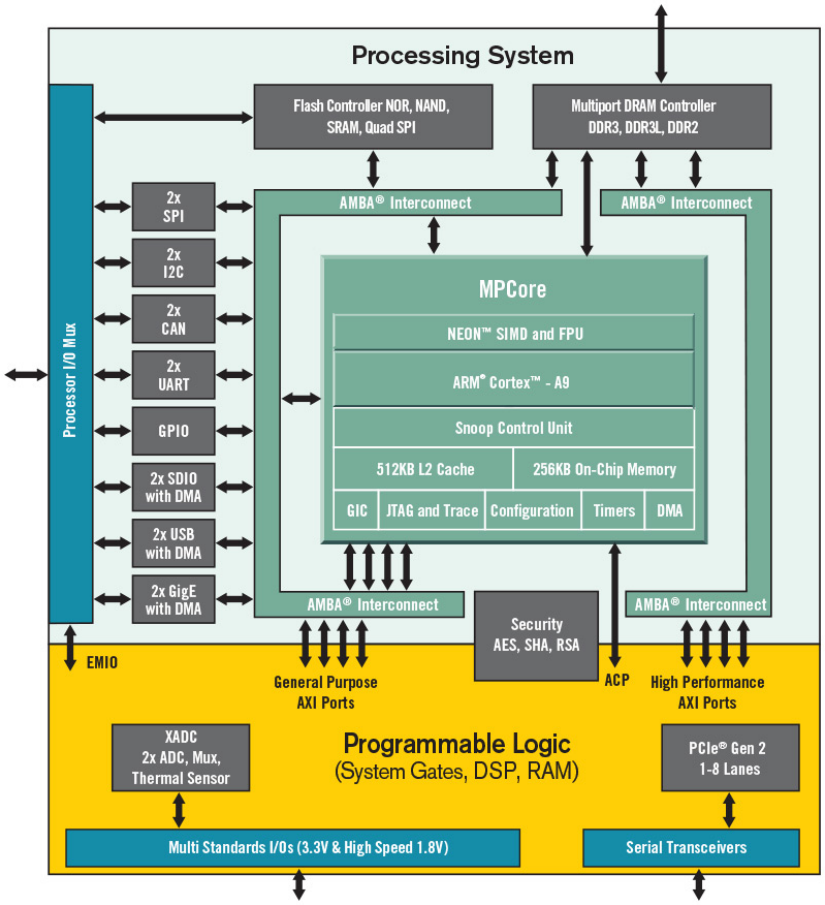


図 2.9　Zynq-7000 の構成 [9]

Zynq-7000 は PS (Processing System) と PL (Programmable Logic) から構成されている。PL は FPGA 領域を示しており、センサコントローラや ADC コントローラ、バッファは PL に実装される。ADC コントローラが出力する 16bit データ信号をネットワークを通して外部 PC に転送するためには、プロセッサがメモリ内のデータを参照する都合上、データを PS にある DDR メモリに内部転送する必要がある。内部転送には DMA 転送というメソッドを採用した。図 2.10 に示しているように、通常データをメモリに書き込んだり読み出したりする作業は、CPU が行う。これを PIO(Programmed I/O) 方式という。対して、DMA(Direct Memory Access) 転送では CPU が DMA コントローラに転送開始の指令のみを与え、メモリアクセスは DMA コントローラが行うため CPU を介

さずにメモリと周辺機器との間でデータを直接転送が可能である。

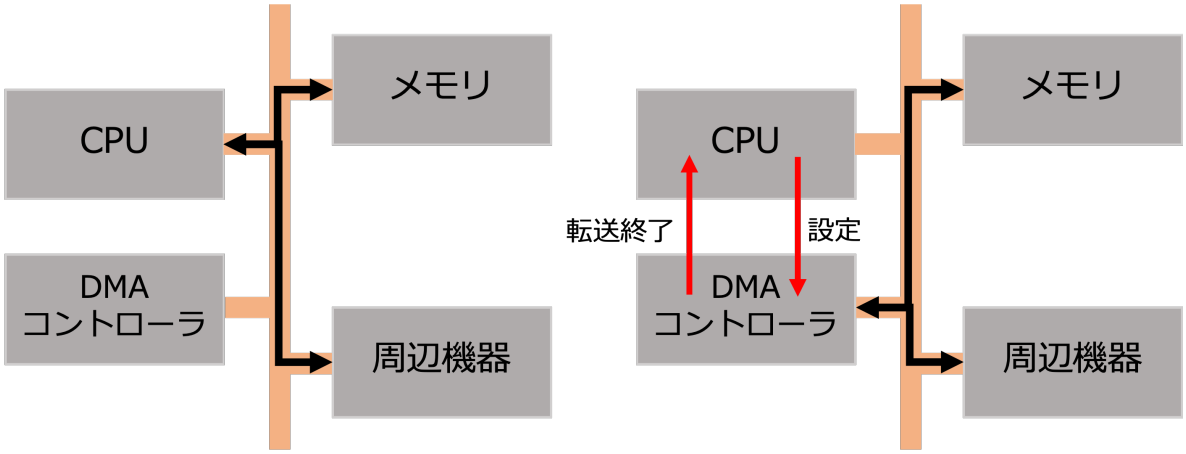


図 2.10　PIO 転送 (左) と DMA 転送 (右)

DMA 転送の前にはセットアップが必要であるため、ADC コントローラが出力するデータを逐次転送していては高速な読出は達成できない。まとまった量のデータを一気に転送すれば、DMA 転送のセットアップにかかる時間の影響を最小限に押さえ、効率的なデータ読み出しが実現可能である。そのため、データを一時的に貯蔵するバッファの実装が必要不可欠である。

バッファの構造は FIFO(First In First Out) を採用した。FIFO とはデータ構造の一種であり、追加した順にデータを読み出す方式のため、待ち行列に例えられる。(図 2.11) 他の処理の準備完了を待つことができるためボトルネックの影響削減効果をデータの順番を崩さずに受けることが可能である。

FIFO の実装と DMA 転送のためには、FIFO に対してのデータを読み書きや、DMA 転送開始のためのトリガー信号の設計、FIFO と DMA コントローラを接続するインターフェースの設計、加えて適切な DMA 転送量と FIFO の容量の見積りが必要である。また、

2.2.4　ネットワーク

メモリに内部転送されたデータを外部 PC に向けて転送するためには、ネットワークを通して外部 PC と通信する必要がある。Eclypse-Z7 では、最大 1Gbps での通信が可能な Giga-bit Eternet が利用可能である。SoC-FPGA の特徴である CPU 機能を活かすことで、Linux OS を搭載し、Linux ライブラリを利用することで Eclypse Z7 と外部 PC 間の通信を行うためのソケットプログラミングを作成する。図 2.12 にソケット通信のフローを示した。ソケットプログラミングとは、ネットワーク上でデータの送受信を可能にするプログラムの一種である。データの送信側 (ホストという、ここでは Eclypse-Z7) と受信

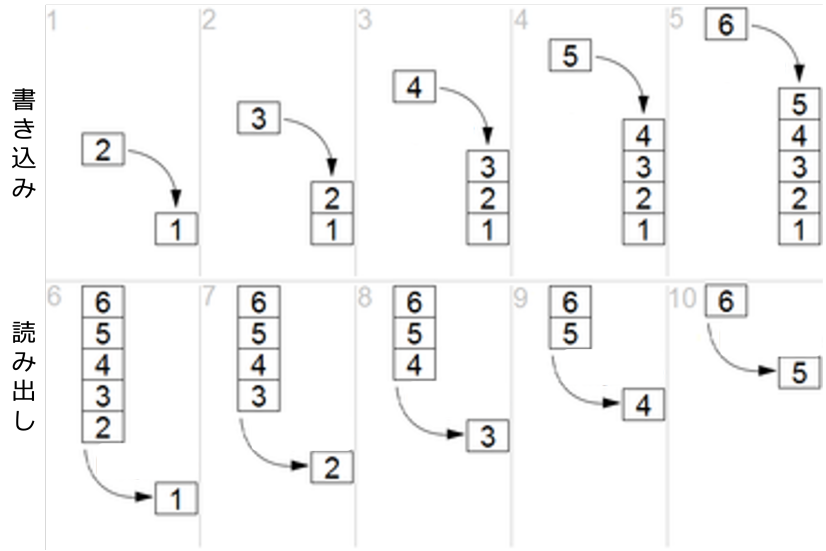


図 2.11　FIFO イメージ図

側 (クライアントという、ここでは外部 PC) の IP アドレスや通信を行うポートで照合を行う。それらの情報をもとに、ホストとクライアントに仮想的な通信の窓口であるソケットを作成し、ソケット間で TCP,UDP に代表される通信プロトコルを用いて通信を行う。

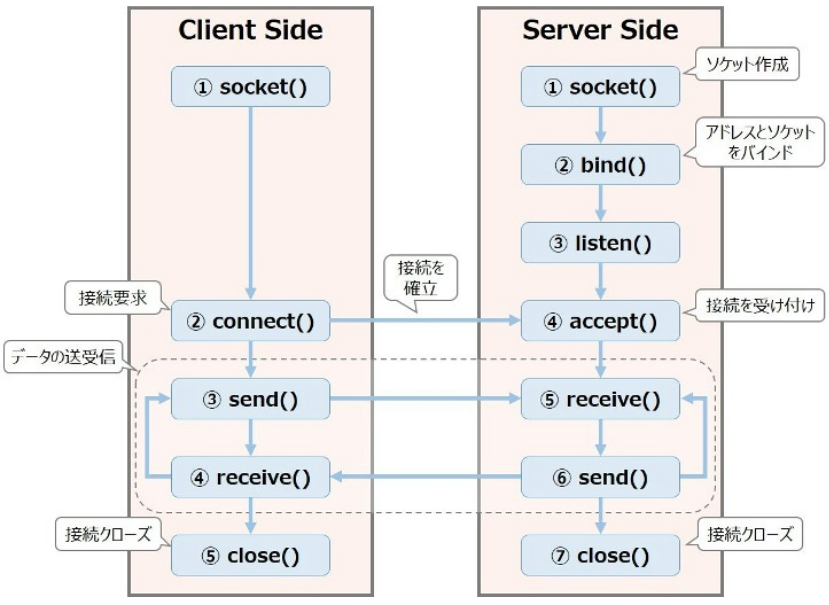


図 2.12　ソケット通信フロー

2.3 開発モジュール

2.2 節で取り上げた開発要素に基づいて、以下に示すモジュールやスキームを設計・開発した。本節では、それらの詳細な動作設計や 2.1.2 節で述べた Vivado の回路シミュレーションの結果を示す。

1. クロック分周器
2. センサコントローラ
3. ADC コントローラ
4. バッファ (FIFO)
5. DMA 転送ロジック
6. DMA 転送速度測定
7. ネットワーク

2.3.1 クロック分周器

図 2.7 の制約を満たすようなセンサコントローラを設計するにあたり、複数周波数のクロック信号生成回路の準備が必要である。センサのデータシートより、クロック信号 (CLK) の範囲は 40KHz から 4MHz までと指定されている。Eclypse-Z7 のデフォルトクロック周波数は 100MHz であるため、そのクロックを分周して 4MHz クロックを用意した。

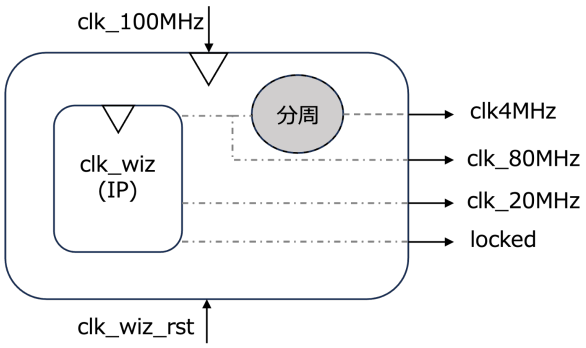


図 2.13　クロック分周器の設計ブロック図

図 2.13 は設計したクロック分周器のブロック図である。信号の説明は表 2.4 に記した。2.1.2 節で紹介した IP である、Clocking wizard (clk_wiz) を用いて 100MHz クロックを分周し、80MHz と 20MHz クロックを作成した (20MHz については後述)。IP の性能により直接 4MHz は生成できないので、100MHz クロックを分周してできた 80MHz ク

ロックを 20 分周することで 4MHz クロックを生成した。locked とは、clk_wiz が正確なクロック信号を生成し、安定して動作をしていることを示す出力信号である。クロック信号が安定しないと他の回路の同期がうまくいかないため、全体の回路に対しての動作開始ロジックとして使用する。

信号名	I/O	説明
clk_100MHz	I	基準クロック信号
clk_4MHz	O	80MHz クロックを分周した 4MHz クロック
clk_20MHz	O	clk_wiz 出力 20MHz クロック
clk_80MHz	O	clk_wiz 出力 80MHz クロック
locked	O	clk_wiz 動作安定信号

表 2.4　クロック分周器の信号

80MHz クロックの 1 周期は 12.5ns であるので、1 周期が 250ns である 4MHz クロックを作成するには 80MHz クロック 10 個毎にレジスタの 0 と 1 を切り替えれば良い (図 2.14)。count[3:0] は 4bit レジスタであり、0 から 16 までの値をカウントすることができる。80MHz クロックに同期させて 0 から 9 までカウントし、$count = 9$ になるたびにレジスタの 0 と 1 を切り替えて 4MHz クロックを作成した。図 2.15 はクロック分周器の回路シミュレーションの様子である。100MHz クロックの入力信号に対し、locked が立った後に 4MHz と 20MHz のクロックが正しく生成されていることがわかる。それぞれの周期は 100MHz(10ns), 80MHz(12.5ns), 20MHz(50ns), 4MHz(250ns) である。

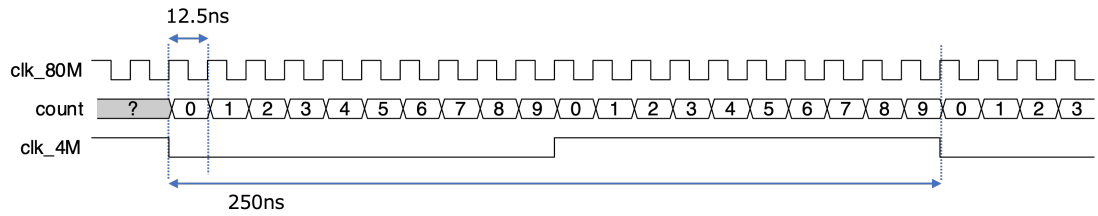


図 2.14　クロック分周方法

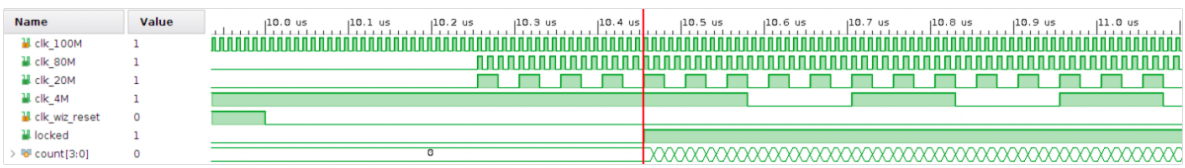


図 2.15　クロック分周器のシミュレーション

2.3.2 センサコントローラ

2.3.1 節で設計したクロック分周器を組み込んで、図 2.7 に沿って設計した。図 2.16 は設計したセンサコントローラのブロック図、表 2.5 は信号の説明である。

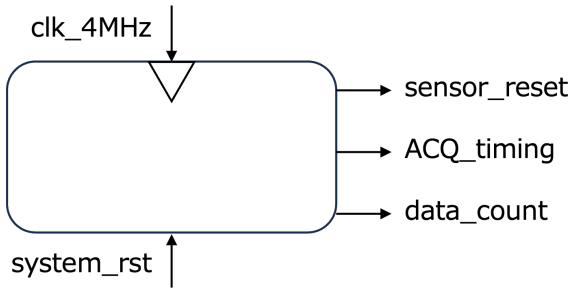


図 2.16　センサコントローラの設計ブロック図

信号名	I/O	説明
clk_4MHz	I	クロック分周器からのクロック信号
system_rst	I	動作リセット信号
sensor_reset	O	センサ動作開始信号
ACQ_timing	O	センサデータ取得タイミング信号
data_count	O	センサ読出データカウント信号

表 2.5　センサコントローラの信号

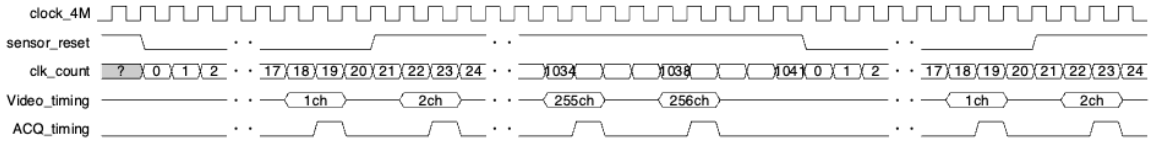


図 2.17　センサコントローラの設計タイミングチャート

図 2.17 は設計上のセンサコントローラの動作を示しているタイミングチャートである。4MHz クロックに同期した clk_count でクロックのパルスを数え、sensor_reset の立ち下がりパルス幅を 21 クロックに設定している。Video_timing は設計の視認性のために Video 信号が出力されるタイミングを示している。ACQ_timing はデータシートより、推奨データ取り込みタイミング (図 2.7 の Trig) に立ち上がるように設計し、後続の回路にてタイミングの指標として用いた。以降、同様の動作を繰り返すために clk_count が

1041 になったらリセットをかけている。シミュレーション結果を図 2.18 に示す。設計段階に加えて、後続の回路やシミュレーションの便利のために、センサのサイクル数を表す cycle_count レジスタを追加している。cycle_count は sensor_reset の立ち下がりで加算している。赤線の直後に sensor_reset が立ち下がり、新たなセンササイクルが開始されている。図 2.19 は次のサイクルへの接続がわかりやすいように編集したものである。センサの 1 サイクルがおよそ 260μs であることが確認できる。

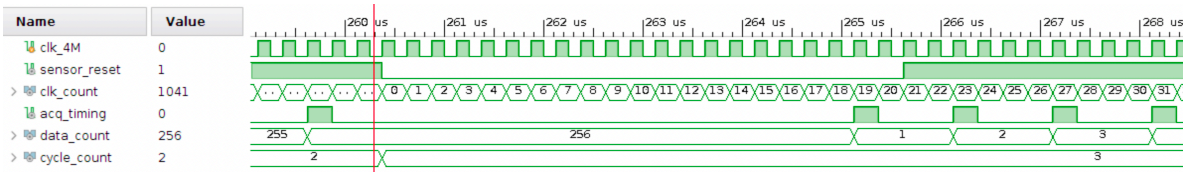


図 2.18　センサコントローラのシミュレーション結果 1

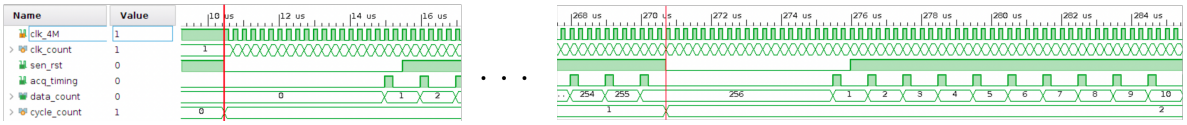


図 2.19　センサコントローラのシミュレーション結果 2

2.3.3　ADC コントローラ

2.2.2 節で述べた通り、ADC コントローラでは SCLK および CS を ADC に供給しつつ、受信する 1bit ずつのデータ信号を 16bit にまとめる必要がある。図 2.20 に設計した ADC コントローラのブロック図を、表 2.6 にそれぞれの信号についての説明を示している。

このコントローラは図 2.21 に示すように内部に 4 つの状態を持つステートマシンとして設計した。図 2.18 より、センサ信号 (Video 信号) は 1μs 毎に出力されるので、4 状態も Video 信号の出力タイミング信号である acq_timing 信号を起点に 1μs 毎に 1 周するような設計になっている。ADC コントローラの 4 状態と ADC の動作の対応関係は以下の通りである。

- S_HOLD アナログデータのサンプリング
- S_FRONT_PORCH データ変換過程への移行期間
- S_SHIFTING データ変換
- S_BACK_PORCH サンプリング過程への移行期間

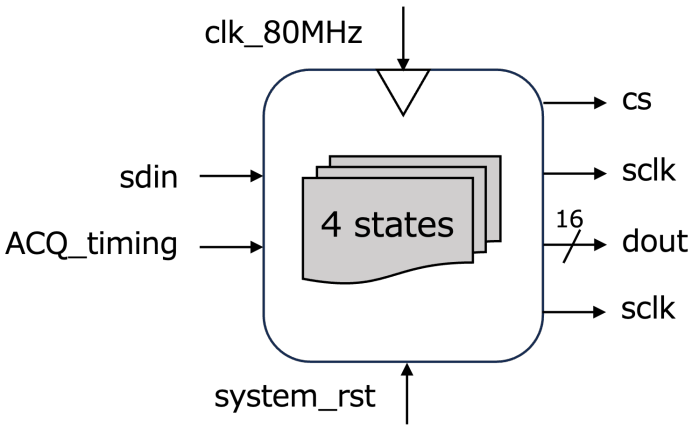


図 2.20　ADC コントローラの設計ブロック図

信号名	I/O	to/from	説明
clk_100M	I	zynq	同期用の 100MHz クロック
rst	I	zynq	ADC コントローラのリセット信号
sdin	I	ADC	ADC の 1bit シリアルデータ信号
cs	O	ADC	ADC の変換プロセス開始信号
sclk	O	ADC	ADC のシリアルデータ (sdin) 送信用同期クロック
drdy	O	バッファ	16bit データの準備完了を示す信号
dout	O	バッファ	16bit データ
acq_timing	I	センサコントローラ	Video 信号出力タイミング信号

表 2.6　ADC コントローラの信号

図 2.22 は設計上の ADC コントローラの動作を示しているタイミングチャートである。ACQ_timing の立ち上がりに同期して内部状態 (AD1_state) のサイクルが開始される。sreg は内部のシフトレジスタであり、sclk に同期して図 2.23 に示すように、1bit ずつのデータをずらしながら詰める工程を合計 16 回行うことで最終的に 16bit データを形成している。この動作は毎回同じであるため、AD1_state=3 の時点で 16bit データは完成している。AD1_state=3 になる時に drdy を立ち上げ、後続の回路に対してデータが利用可能であることを伝えている。

図 2.24 は、以上のことを踏まえて設計した ADC コントローラのシミュレーション結果である。

ACQ_timing の立ち上がりで内部状態 (AD1_state) が遷移を始め、AD1_state=0 になる。この間に ADC のサンプリング時間分待機して AD1_state=1 に遷移し、その後 1 クロック分待って AD1_state=2 に遷移し、sclk に同期して 1bit ずつのデータを sreg

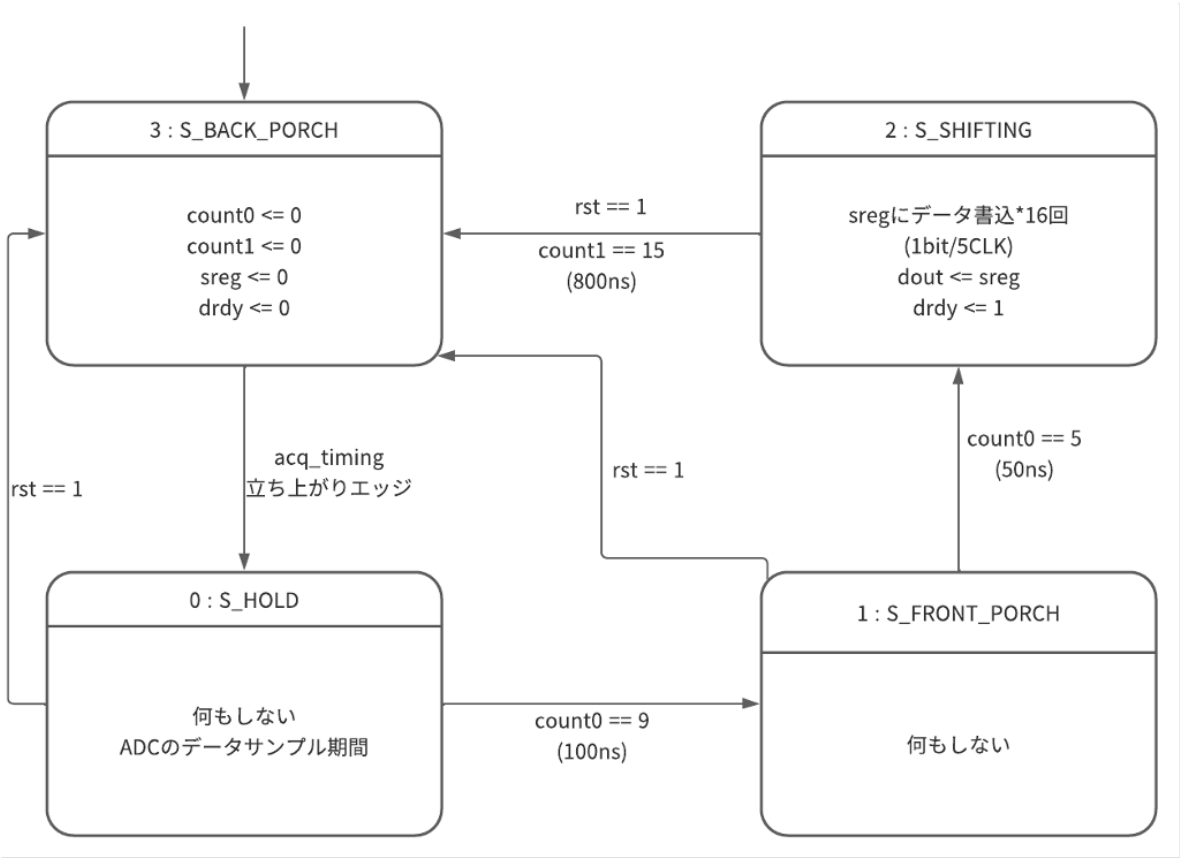


図 2.21　ADC コントローラの状態遷移図

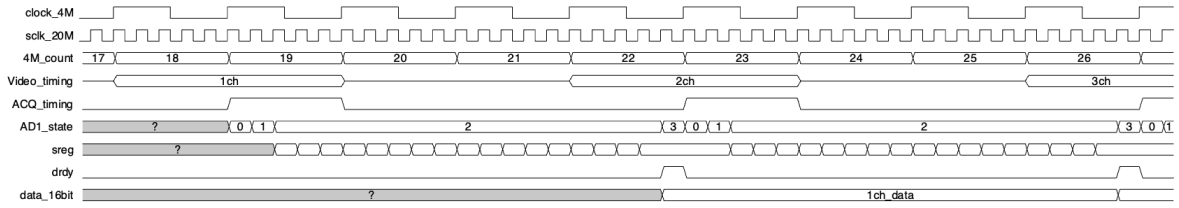


図 2.22　ADC コントローラの設計タイミングチャート

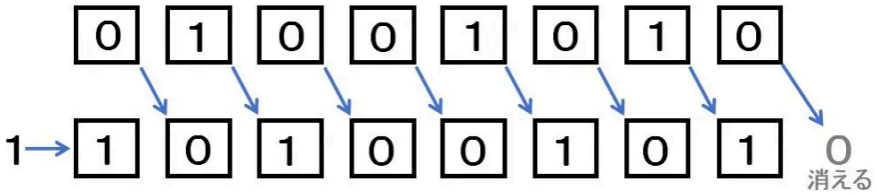


図 2.23　シフトレジスタの動作

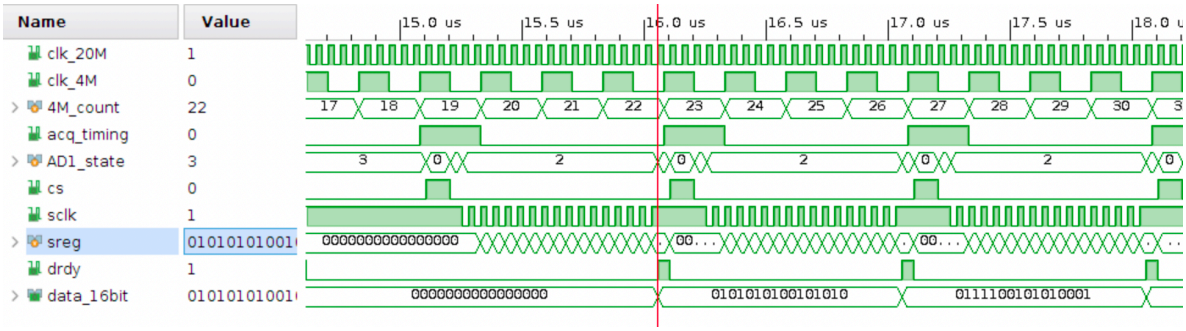


図 2.24　ADC コントローラのシミュレーション結果

に詰めている。dout(data_16bit) の準備ができたら、sreg の値を出力用のレジスタに代入し、AD1_state=3 に遷移して drdy 信号を有効にし、sreg のリセットを行い、次の acq_timing の立ち上がりを待っている。これら一連の状態遷移が 1μs 毎に行われていることが確認できた。

2.3.4　バッファ

2.2.3 節で紹介したように、ADC コントローラが出力する 16bit データを一時的に保管するバッファを設計した。便宜上本研究では FIFO と呼称する。図 2.25 に設計した FIFO のブロック図を、表 2.7 にそれぞれの信号の説明を示している。FIFO はパラメータとして幅・深さを持つ。読み書きしたいデータに合わせて幅を、FIFO が溢れないような深さを設定する必要がある。適切な深さを決定するためには、FIFO 書き込みレートと FIFO 読出速度、つまり DMA 転送速度を測定する必要があるため、設計段階では FIFO の幅のみ決定し、深さについては決定していない。

ADC データが 16bit なのに対し、FIFO データ幅を 17bit に設定している。2.3.5 節で詳しく紹介するが、FIFO と DMA モジュール間の通信は AXI(Advanced eXtensible Interface) に則る必要がある。AXI インターフェースには複数の種類が存在するが、本研究ではストリーミングデータに適した AXI Stream を採用している。AXI Stream では、転送するデータサイズを指定した一括転送が特徴だが、転送するデータの末尾を示す信号が要求されている。そのため、16bit データに加えてデータ末尾情報を追加して FIFO データ幅を 17bit に設定した。

また、本 FIFO は書込領域と読出領域で異なる周波数のクロックに同期していることが特徴として挙げられる。書込側クロックは ADC コントローラの 20MHz クロックで決定されているが、読出側クロックに対する制約はないためである。読出側クロックの速度は DMA 転送速度に直結するため、Eclypse-Z7 の標準クロックである 100MHz が理想であるが、クロック比が整数倍にならず位相がずれてしまい意図しない動作につながるため、

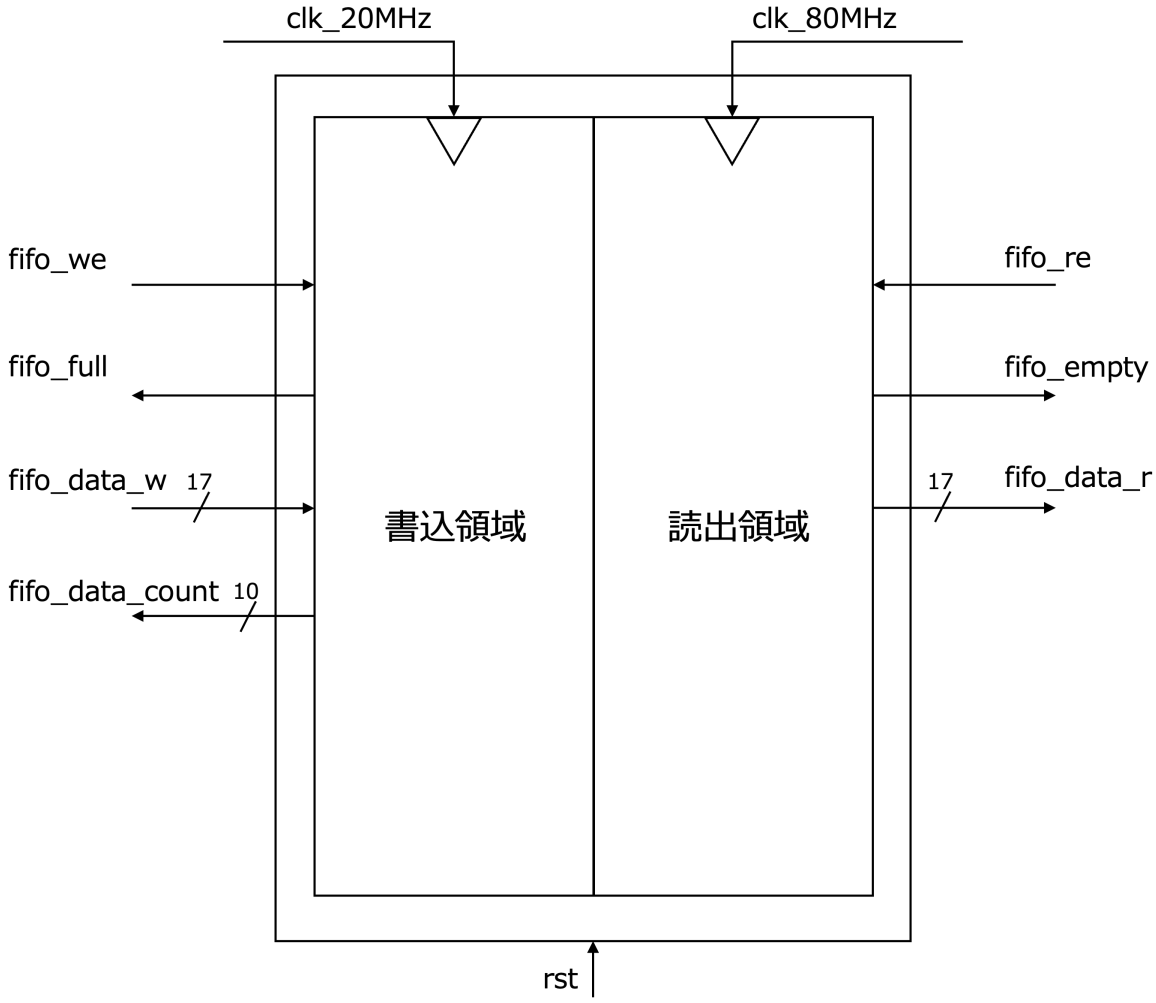


図 2.25　FIFO の設計ブロック図

読出側クロックは 80MHz に設定した。

図 2.26 に基本的な FIFO の動作を示した。書込トリガー (write_enable) が立ち上がっている間に経過したクロック (write_clk) の数分、データを書き込む (①, ②)。読出も同様にして、読出トリガー (read_enable) が立ち上がっている間に経過したクロック (read_clk) の数分、データを読み出す (③, ④)。書込・読出に応じて FIFO 内部のデータ数 (fifo_data_count) が変動し、FIFO 内にデータがないときは FIFO 空フラグ (fifo_empty) が立ち上がっていることが確認できる。図には示していないが、FIFO 内のデータ数が FIFO 深さを超過した際には FIFO 満杯フラグが立ち上がる。

図 2.27 に FIFO 書込のシミュレーション結果を示した。①にてセンサコントローラが acq_timing を立ち上げ、それを受けて ADC コントローラが動作を開始する。②で ADC コントローラは 16bit データを出力する。①では ADC コントローラの drdy 信号

信号名	I/O	説明
clk_20MHz	I	書込同期用クロック
clk_80MHz	I	読出同期用クロック
rst	I	FIFO クリア用のリセット信号
fifo_we	I	FIFO 書込トリガー (write_enable)
fifo_re	I	FIFO 読出トリガー (read_enable)
fifo_full	O	FIFO 満杯フラグ
fifo_empty	O	FIFO 空フラグ
fifo_data_w	O	FIFO 書込データ
fifo_data_r	O	FIFO 読出データ
fifo_data_count	O	FIFO 内データカウント

表 2.7　FIFO の信号

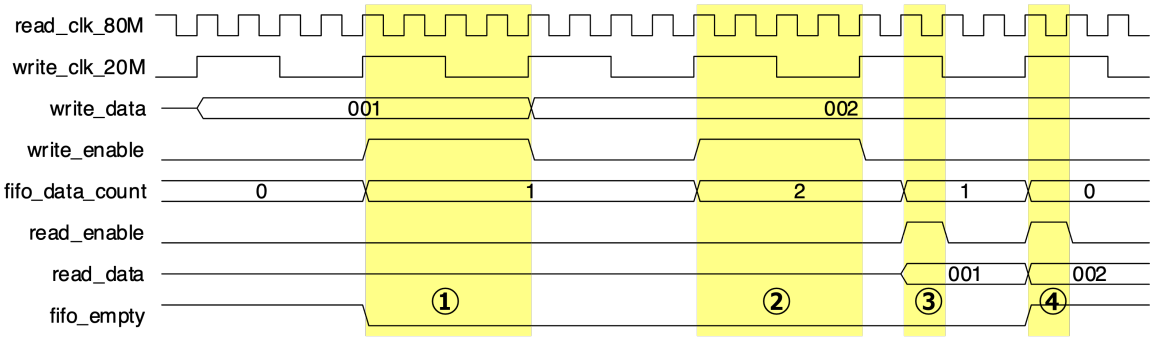


図 2.26　FIFO の動作方法

を受けて fifo_we が立ち上がり、FIFO へのデータ書込が行われている。EOL(End of Line) 信号はデータ末尾情報を示す信号であり、data_count=256 の時に 1 になるように設計している。そのため、③時点での fifo_data_w では 16bit データ (ad1_data_out) の先頭に 0 が追加されている。

実際に EOL 信号が 1 になる時の様子は図 2.28 に示した。data_count=256 になったタイミングで EOL=1 になっている。①は 256 個目の FIFO 書込データであり、先頭のデータ末尾情報が 1 になっている。その前に既に EOL=1 がデータに付与されているが、図 2.26 で説明した通り、FIFO 書込が行われるのは書込トリガー (fifo_we) が立ち上がっている間に経過したクロック (clk_20M) の数分であるため、実際に書き込まれるデータは②時点のものである。以上のようにして FIFO 書込動作が正常に行われていることを確認した。FIFO 読出については、DMA 転送と直接関わっているため 2.3.5 節で述べる。

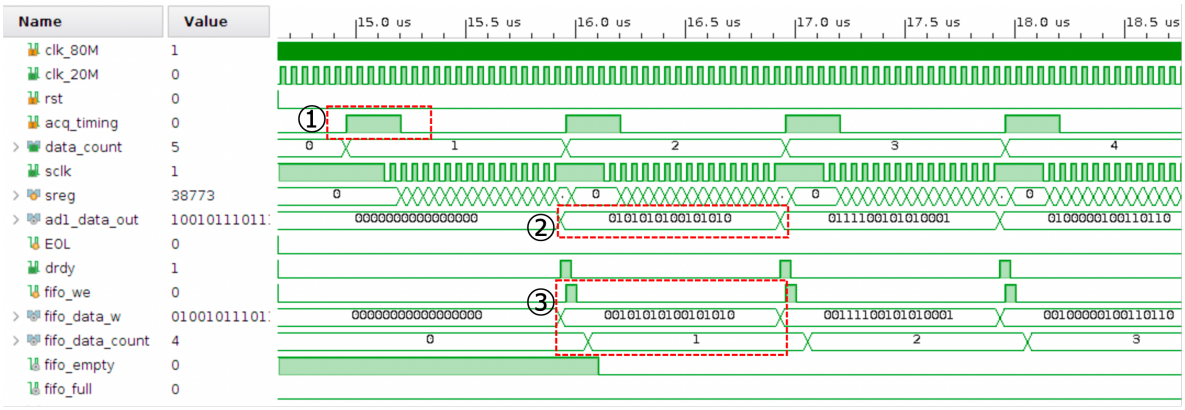


図 2.27　FIFO 書込シミュレーション結果 1

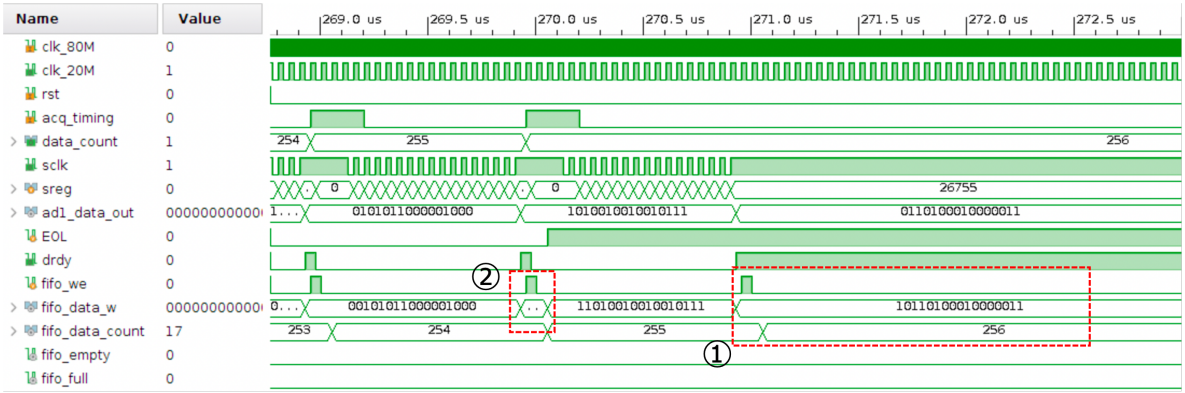


図 2.28　FIFO 書込シミュレーション結果 2

2.3.5　DMA 転送ロジック

FIFO 内データをメモリに内部転送するために、IP コアである AXI DMA を用いた。図 2.29 は AXI DMA のブロック図である。FIFO から読み出されたデータは AXI Stream(S2MM) インターフェースを通して DMA コア内部のバッファである AXI Datamover(S2MM) に格納され、AXI Memory Map Write(S2MM) を通してメモリへ転送される。AXI 規格では、PS を MM(Memory Map), PL を S(Slave) と表現しているため、S2MM(Slave to Memory Map) は PS へ向けた通信を意味している。他にも様々なブロックがあるが、今回使用するのはメモリ書込であるため、メモリ読出 (MM2S) は使用しない。AXI Stream インターフェースは、基本的に表 2.8 に示す 4 つの信号からなる。その他オプション信号があるが、今回は使用していない。図 2.30 に AXI Stream において、data1-data6 の 6 つのデータを転送する場合のハンドシェイク (通信方法) を示した。Slave 側 (データを受け取る側) は、データの受け入れが可能なときに TREADY

を立てる。この間、TVALID が立っている間のデータ (data1-data6) が有効データとして認識される。data6 はこの転送における最終データであるので、data6 を転送している間 TLAST が立っている。

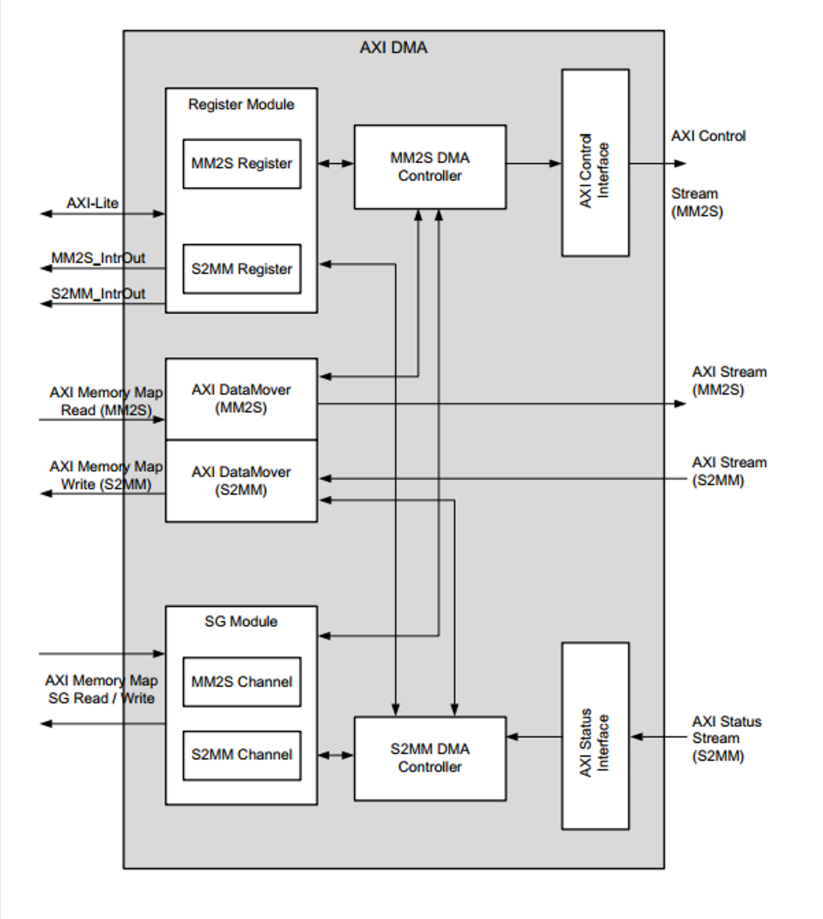


図 2.29　AXI DMA のブロック図

[8]

信号名	信号ソース	説明
AXIS_TVALID	Master	Master データが有効であることを示す
AXIS_TREADY	Slave	Slave がデータ転送可能であることを示す
AXIS_TLAST	Master	Master データが最終ワードであることを示す
AXIS_TDATA	Master	Master データ

表 2.8　AXI Stream 信号

以上のことを踏まえ、FIFO と DMA コアの間の AXI Stream 接続を表 2.6 のように

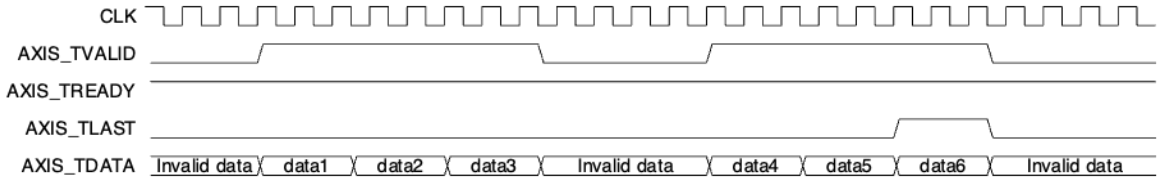


図 2.30　AXI Stream ハンドシェイク

設計した。表 2.8 からわかるように、TREADY のみ Slave 側 (ここでは DMA コアが相当) が出力する信号である。Vivado の機能シミュレーションでは、ZYNQ やその他のハードウェアを含んだシミュレーションができない。FIFO 読出シミュレーションとして、256 個のデータが FIFO に書き込まれた後に FIFO 書込を停止し、手動で TREADY を立ててデータ読出を確認した。図 2.31 はそのシミュレーションの様子である。AXI Stream 信号および fifo_re は表 2.9 の通りである。80MHz クロックに同期して FIFO 内データが読み出され、AXIS_TDATA に適用されていること、AXIS_TVALID および AXIS_TLAST が設計どおりに動いていることが確認できた。

信号名	bit 幅	設計	説明
TVALID	1	~fifo_empty	FIFO が空でない
TDATA	16	fifo_data_r[15:0]	FIFO 出力 (17bit) の下位 16bit
TLAST	1	fifo_data_r[16]	FIFO 出力 (17bit) の上位 1bit
fifo_re	1	TREADY&TVALID	DMA コアが転送可能かつ FIFO が空でない

表 2.9　FIFO と DMA コアの接続

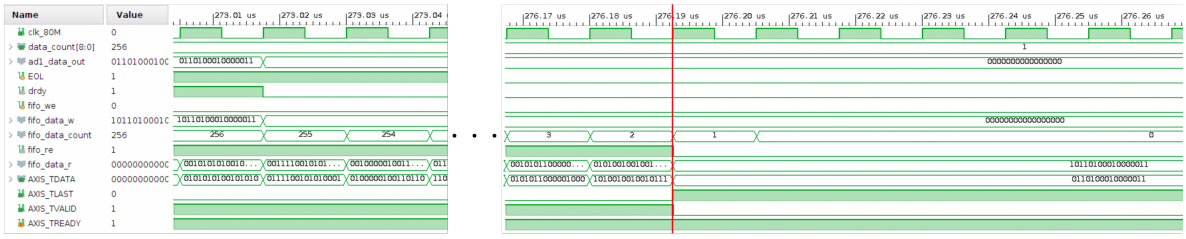


図 2.31　FIFO 読出シミュレーション結果

2.3.6　DMA 転送速度測定

2.3.5 節・2.2.3 節で述べた通り、バッファの適切な深さおよび適切な DMA 転送サイズを決定するために DMA 転送速度を実機を用いて測定した。

2.3.6.1　DMA 転送の制御レジスタ

図 2.10 にあるように、DMA 転送を実行するためには DMA コントローラの設定が必要である。図 2.32 は DMA コアのレジスタ空間である。DMA コアのベースアドレスからのオフセットに、それぞれ役割を持たせたレジスタが設計されている。

DMA コアは、CPU によってメモリマップド空間内に配置され、CPU からは FPGA 内のメモリアドレスとしてアクセスが可能である。CPU が制御レジスタや設定レジスタなどの内部レジスタにアクセスして値の書込を行うことで DMA 転送を開始する。2.3.5 節で述べたように、今回の DMA 転送は S2MM チャンネルを用いている。ここで、図

アドレス空間オフセット[1]	名前	説明
00h	MM2S_DMACR	MM2S DMA 制御レジスタ
04h	MM2S_DMASR	MM2S DMA ステータス レジスタ
08h ～ 14h	予約	N/A
18h	MM2S_SA	MM2S ソース アドレス
1Ch ～ 24h	予約	N/A
28h	MM2S_LENGTH	MM2S 転送長さ (バイト)
30h	S2MM_DMACR	S2MM DMA 制御レジスタ
34h	S2MM_DMASR	S2MM DMA ステータス レジスタ
38h ～ 44h	予約	N/A
48h	S2MM_DA	S2MM デスティネーション アドレス
4Ch ～ 54h	予約	N/A
58h	S2MM_LENGTH	S2MM バッファー長さ (バイト)

図 2.32　AXI DMA レジスタのアドレスマップ

2.32 の S2MM 関連レジスタを見ていくことにする。32bit のレジスタのうち、どのレジスタがどの制御に関与するかを予め設計されている。これをアドレスマップという。図 2.33 は、S2MM チャンネルの DMA 制御レジスタ、S2MM_DMACR である。特に重要なのが、S2MM_DMACR[0]; RS である。この bit フィールドは読み書きどちらも可能であり、CPU が書き込みを行うことで DMA チャンネルの実行と停止を制御している。

0: **停止**　現在進行中の DMA 転送がある場合は、その転送の完了後に DMA が停止される。
1: **実行**　DMA 動作を開始する。

図 2.34 は、S2MM チャンネルの DMA ステータスレジスタ、S2MM_DMASR である。このレジスタはステータス参照のためのものであり、読み込み専用になっている。主に S2MM_DMASR[0]; Halted, S2MM_DMASR[1]; Idle を参照して用いる。

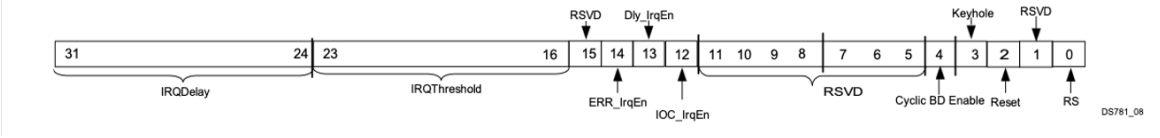


図 2.33　S2MM_DMACR レジスタのアドレスマップ

Halted=0　DMA チャンネル動作中
Halted=1　DMA チャンネル停止中
Idle=0　DMA 転送サイクル実行中、アイドルでない
Idle=1　DMA 転送サイクル完了、アイドル

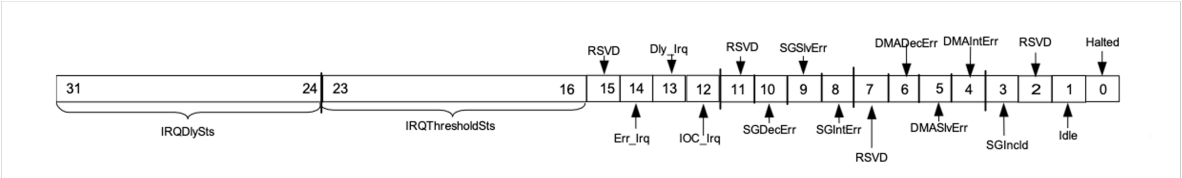


図 2.34　S2MM_DMASR レジスタのアドレスマップ

2.3.6.2　PYNQ

前節で述べた通り、DMA 転送の実行には DMA 制御レジスタの読み書きが不可欠である。そこで、ソフトウェアでのコントローラの設計が必要になる。この要件に対して、PYNQ(Python Productivity for Zynq) が有力である。PYNQ とは、Zynq シリーズ FPGA 向けのオープンソースプロジェクトである。Python を使用して FPGA デザインやアプリケーション開発が可能であり、スクリプト内で FPGA の設定やプログラムのロード、ひいては DMA などの IP コアにアクセスするライブラリが提供されている。これにより FPGA デザインとソフトウェアの統合がスムーズになる。

しかし、Eclypse-Z7 に PYNQ を実装できなかったため、Eclypse-Z7 と同様の SoC-FPGA チップである Zynq-7000 を搭載した PYNQ-Z1 ボードをお借りして DMA 転送速度を計測した。PYNQ ライブラリの DMA モジュールを用い、回路内の DMA コアをソフトウェア上でバインドし、コントロールレジスタに書き込んだりステータスレジスタを読み込んだりしている。

2.3.6.3　**測定用回路**

DMA 転送速度を測定するために、 32bit(4Byte) 乱数生成器を用いた。測定方法は以下の通りである。

1. 作成する乱数サイズの決定 (4N [Byte])
2. 乱数の生成
3. FIFO 書込
4. 2. 3. を 1. のサイズに達するまで繰り返す
5. 乱数生成器の停止
6. DMA コア内バッファに FIFO データ全てを読み出す (4N [Byte])
7. タイムスタンプ取得 (転送開始前時刻)
8. DMA 転送 (4N [Byte])
9. DMA 転送終了確認
10. タイムスタンプ取得 (転送終了後時刻)
11. 2. から 10. までを 10000 回繰り返す

データサイズを 10KB から 500KB まで 10KB 刻みに合計 50 個取得した。図 2.35 に示すように、典型的な時間が 2 つ確認できる。これは、DMA 転送の所要時間に 2 つの要因があるからである。

DMA 転送セットアップ　転送サイズ、転送先メモリアドレスの設定にかかる時間
バッファ転送　実際にデータ転送にかかる時間

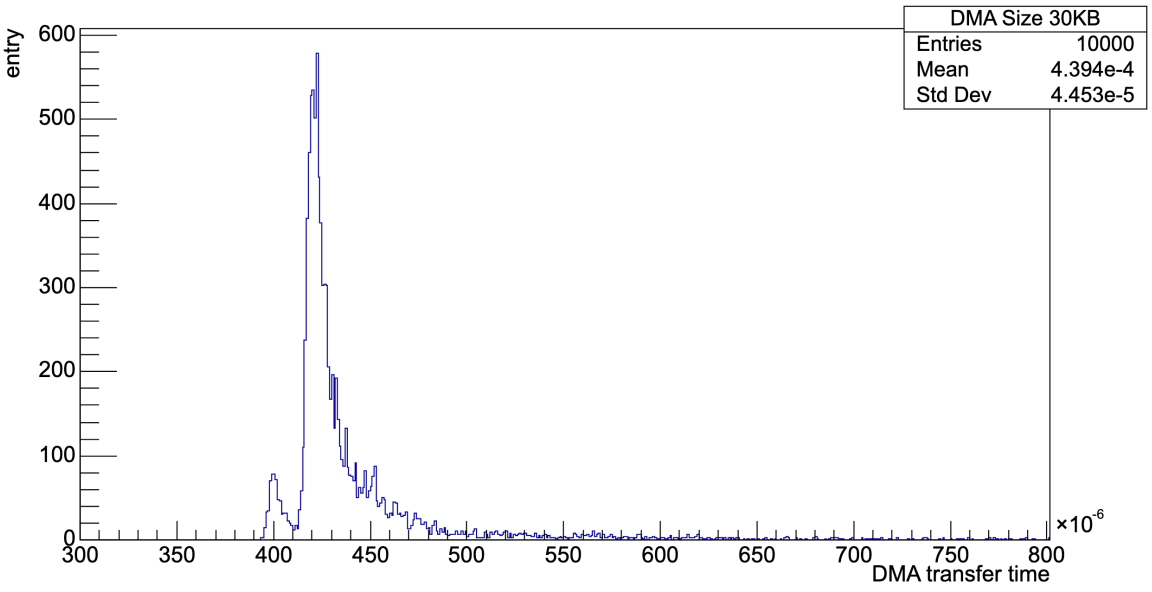


図 2.35　DMA 転送時間：30KB

図 2.36 は、データサイズを横軸に、所要時間を縦軸において 2 次元表示したものである。このグラフから、DMA 転送速度が求められる。本検証ではおおよそ 300MBps という結

果を得た。DMA 転送セットアップにかかる時間は、データサイズに関わらず決定され、データ転送にかかる時間はデータ量に比例して増加する。しかし、DMA 転送セットアップに関してユーザーは直接関与できず、一定の割合でばらつきが生じる。今回の検証ではおおよそ 1% 程度の確率で大きく所要時間がかかる転送が確認できた。DMA 転送速度の理論値は、同期クロックが 100MHz であり、バッファアドレス幅が 32bit(4Byte) であるため、400MBps と計算できる。DMA コアの仕様書には、S2MM チャンネルの実計測総スループットは 298.59MBps と記載されており、本検証と一致する。

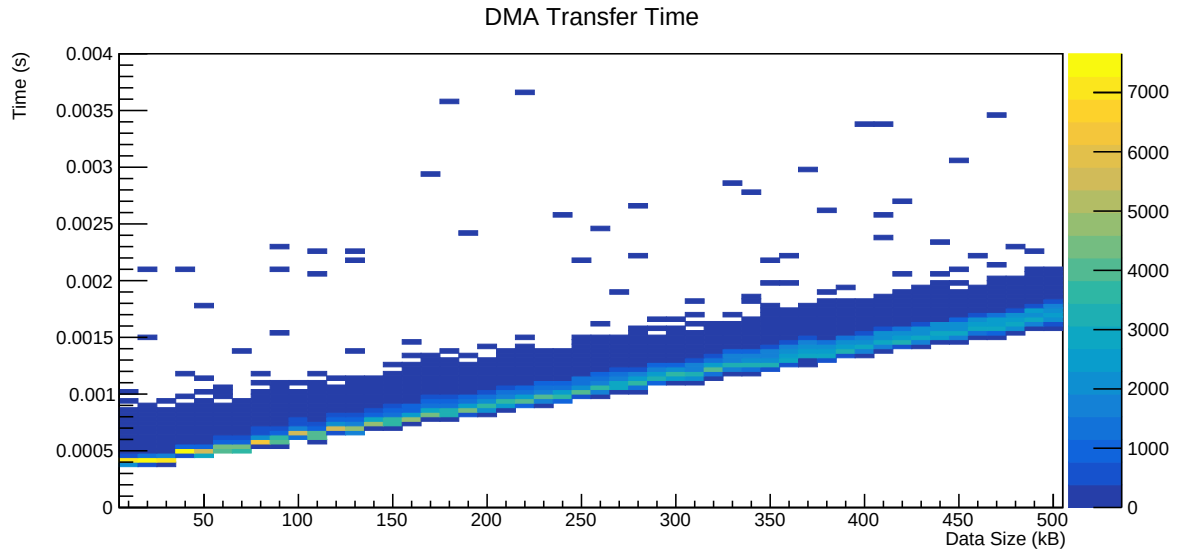


図 2.36　データサイズ別に DMA 転送時間を 2 次元表示したもの

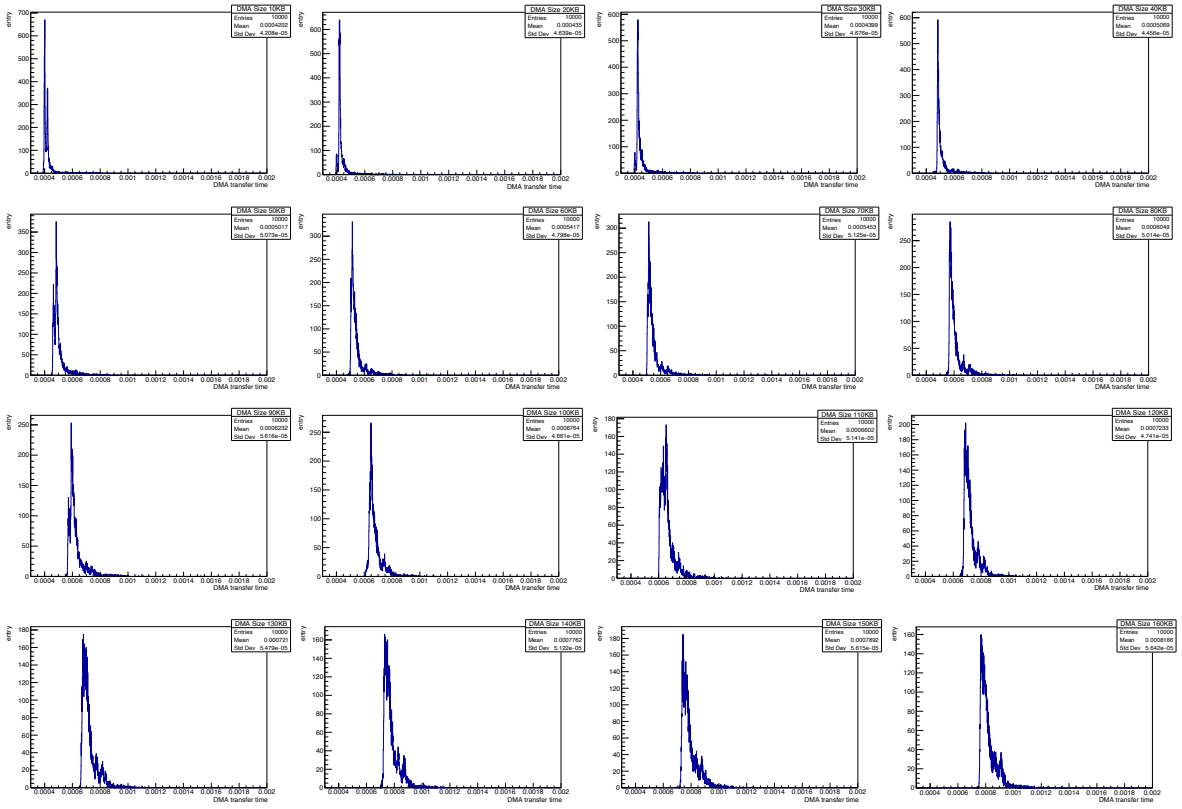


図 2.37　DMA 転送時間：10KB-160KB

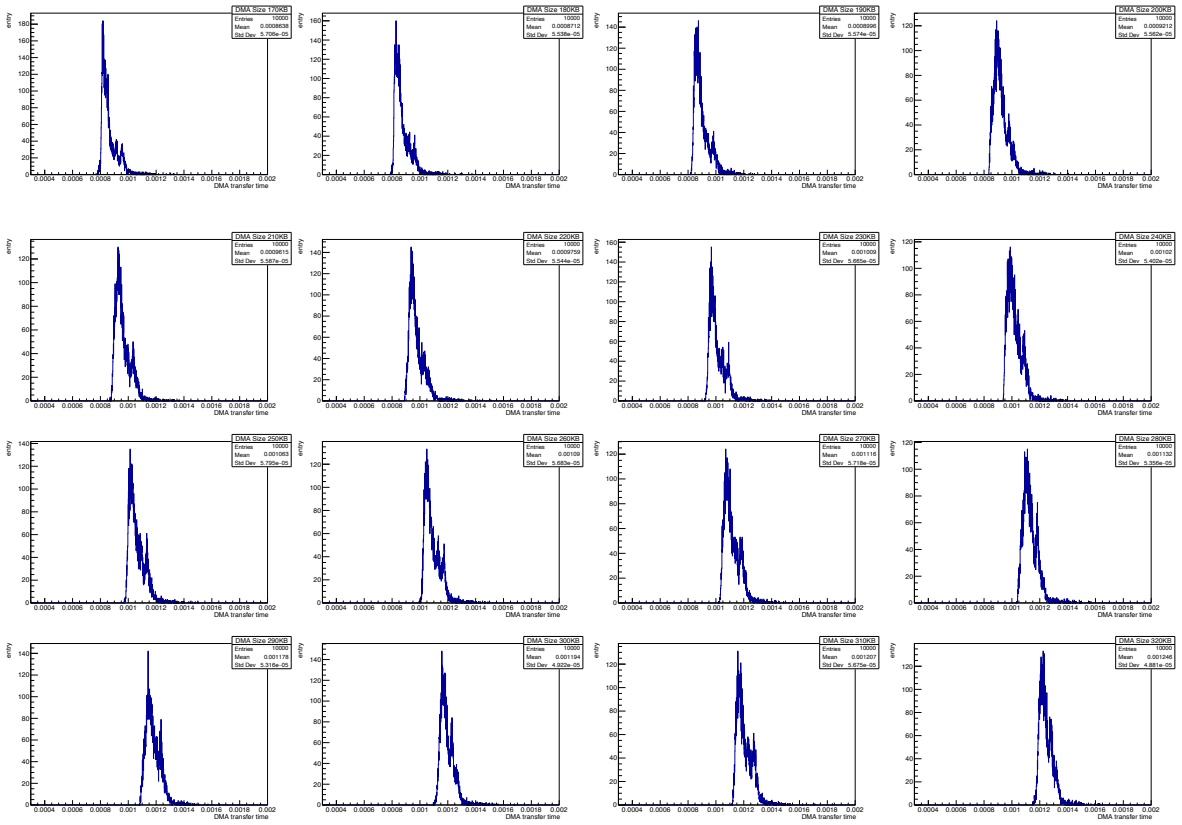


図 2.38　DMA 転送時間：170KB-320KB

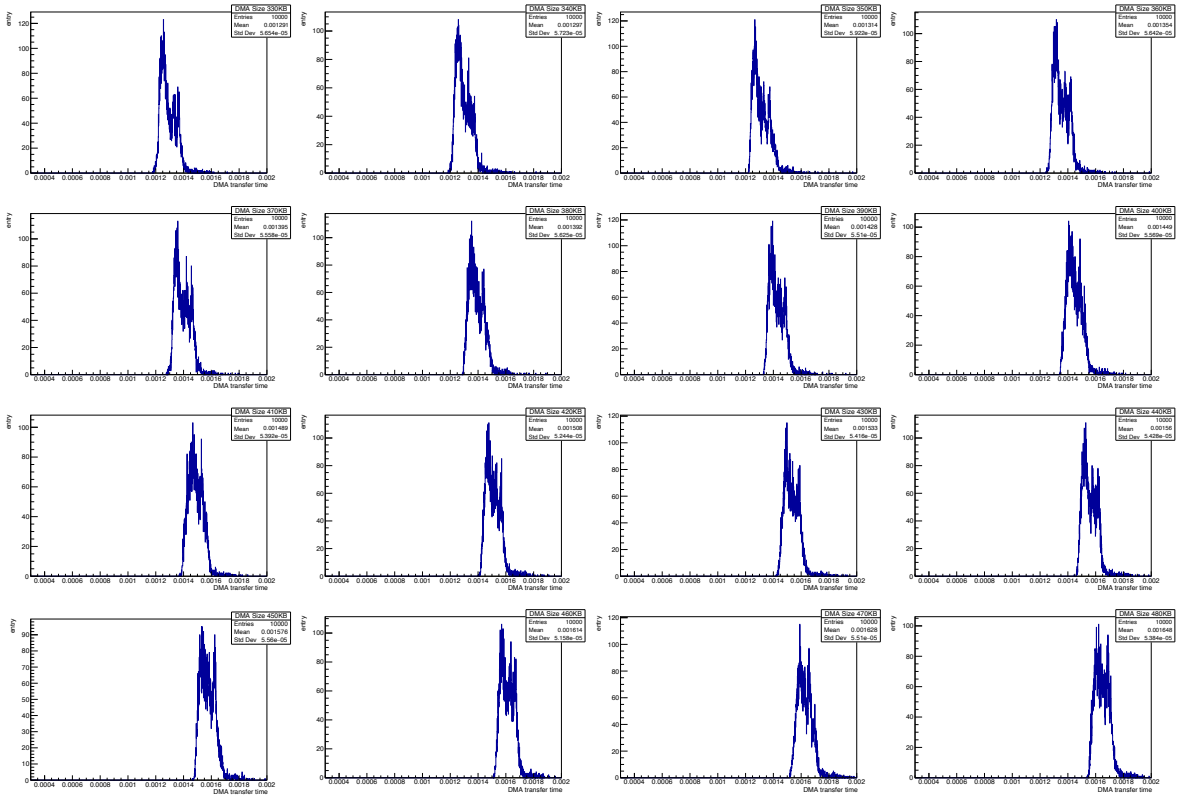


図 2.39　DMA 転送時間：330KB-480KB

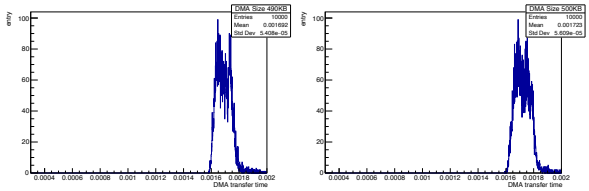


図 2.40　DMA 転送時間： 490KB-500KB

2.3.7 ネットワーク

2.2.4 節で述べた、ネットワークプログラムを設計した。図 2.41 に示すように、データ送信側 (PYNQ-Z1) とデータ受信側 (外部 PC) でそれぞれ送信用、受信用のプログラムを作成した。2.3.6.3 節で用いた乱数生成回路の出力をネットワークを通して転送し、ネットワーク速度測定を行った。送信側・受信側フローを以下に示す。送信側フロー

1. 送信元・送信先の IP アドレスとポートを設定する
2. 通信プロトコルを UDP に指定してソケットを作成し、送信先ソケットにバインドする
3. メモリに転送されたデータを送信先ソケットに送信する

受信側フロー

1. 送信元 IP アドレスとポートを設定する
2. 通信プロトコルを UDP に指定してソケットを作成し、送信元ソケットにバインドする
3. 常に受信待ちループをさせ、受信データをファイル書込する

図 2.42 は送信側 Python スクリプトを実行している Jupyter Notebook の画面、図 2.43 は受信側でのコンソールである。ネットワークを通してデータの転送に成功した。この場合、32bit 乱数 1024 個からなる 4096Byte データの転送に 83.65 秒かかっており、転送速度は約 4Mbps となった。ただし、所要時間には乱数生成時間が含まれているため、この転送速度は正確な性能を示していない。その後、転送データを 2048Byte の固定文字列 1000 個からなる 2048KByte に置き換えたところ、総所要時間は 17332950 ナノ秒であり、正味の転送速度は式 2.2 よりおよそ 1Gbps であることが確認できた。

$$\frac{2048000}{1733295}[GBps] \simeq 120MBps = 960Mbps \simeq 1Gbps \tag{2.2}$$

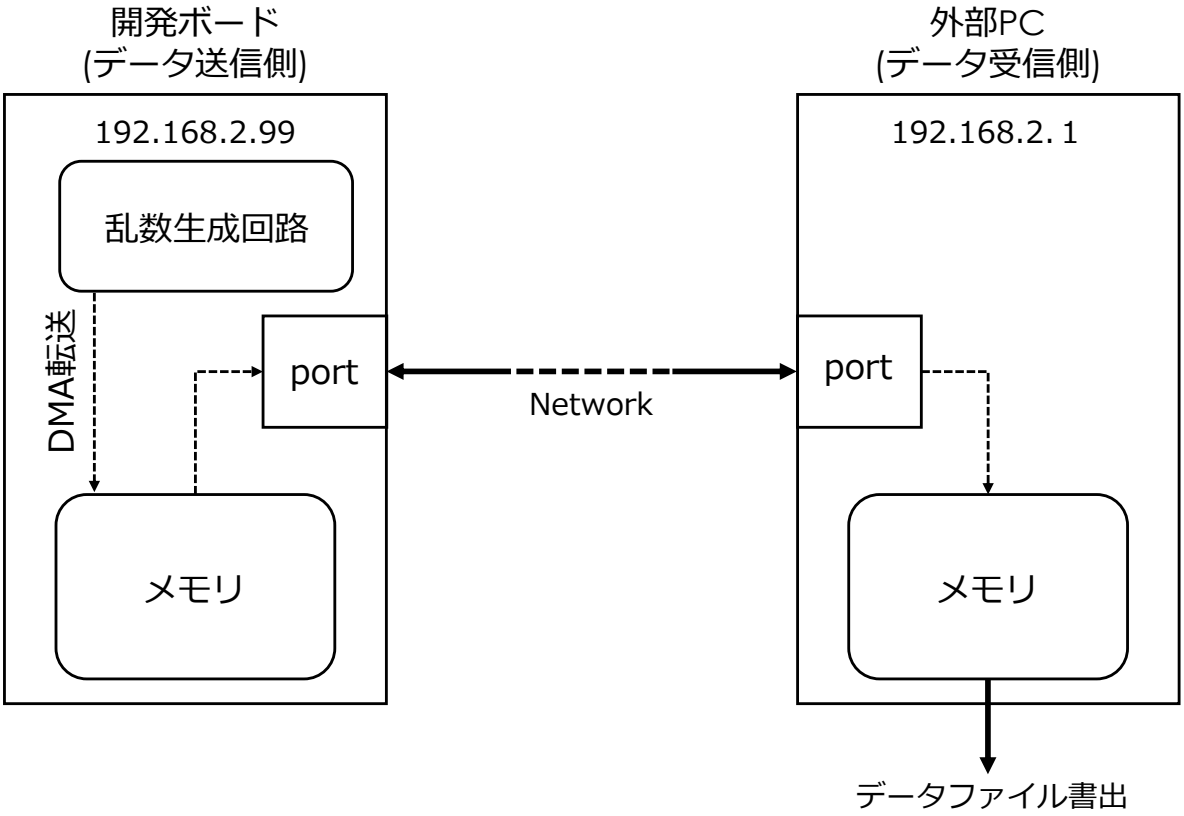


図 2.41　作成したネットワークプログラム

In [10]:

```
from pynq import Overlay
from pynq import allocate
import socket
import numpy as np
import time

pl = Overlay("trng_ip.bit")
rng = pl.TRNG_IP_0
dma = pl.axi_dma_0
buffer = allocate(shape=(1024,), dtype=np.uint32)

SrcIP = "192.168.2.99"
SrcPort = 11111
SrcAddr = (SrcIP, SrcPort)
DstIP = "192.168.2.1"
DstPort = 22222
DstAddr = (DstIP, DstPort)
ClientSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
ClientSock.bind(SrcAddr)

time_start = time.time()

for i in range(10000):
    rng.parameter = i
    rng.start(4096)
    dma.recvchannel.transfer(buffer)
    rng.wait()
    dma.recvchannel.wait()
    #print("ID: {0}, Data: {1:08x}".format(i, buffer[0]))
    print("ID: {0}, Data: {1}".format(i, buffer))
    rng.stop()
    data = buffer.tobytes()
    sent = ClientSock.sendto(data, DstAddr)
    buffer.freebuffer()

time_end = time.time()
time_total = time_end - time_start
print("Time: {0} [sec]" .format(time_total))
```

```
ID: 9982, Data: [2147483647 4294967295 4294967295 ... 4294967295 4294967295 4294967295]
ID: 9983, Data: [4294967295 4294967295 4294967295 ... 4294967295 4294967295 4294967295]
ID: 9984, Data: [4294967279 4292868092 3741319151 ... 1073741856          0   33562625]
ID: 9985, Data: [ 801112063 2147221503 4290772991 ... 2553070800   81815546  819720060]
ID: 9986, Data: [4294965249 4190143045   67108992 ...          0          0          0]
ID: 9987, Data: [ 709711995 4294934527 4294967295 ... 4294967295 4294967295 4294967295]
ID: 9988, Data: [2139390000        768    2098053 ... 1803860782 3186528767 2787215490]
ID: 9989, Data: [1822767347 4021825984    2101388 ...    3670784 1575191040       1088]
ID: 9990, Data: [1073741824          8          0 ...  134545472  193331232 2148036618]
ID: 9991, Data: [4294572959 3196054462 3983198206 ...          0          0          0]
ID: 9992, Data: [   3211262 1564657526 2137062717 ... 4294966255 1270478337 1343848446]
ID: 9993, Data: [4294967295 4294967295 4294967295 ... 2519536460  946943000   12728352]
ID: 9994, Data: [0 0 0 ... 0 0 0]
ID: 9995, Data: [4294967295 4294967295 4294967295 ... 4294967295 4294967295 4294967295]
ID: 9996, Data: [1610612736          0          0 ... 4191665399 4227295267 1807339731]
ID: 9997, Data: [2013265919 4294967295 4294967295 ... 4294967295 4294967295 4294967295]

ID: 9998, Data: [0 0 0 ... 0 0 0]
ID: 9999, Data: [ 521592807 4294967294 2143289343 ... 4294967295 4294967295 4294967295]
Time: 83.64626550674438 [sec]
```

図 2.42　PYNQ-Z1 上の Jupyter Notebook スクリプト

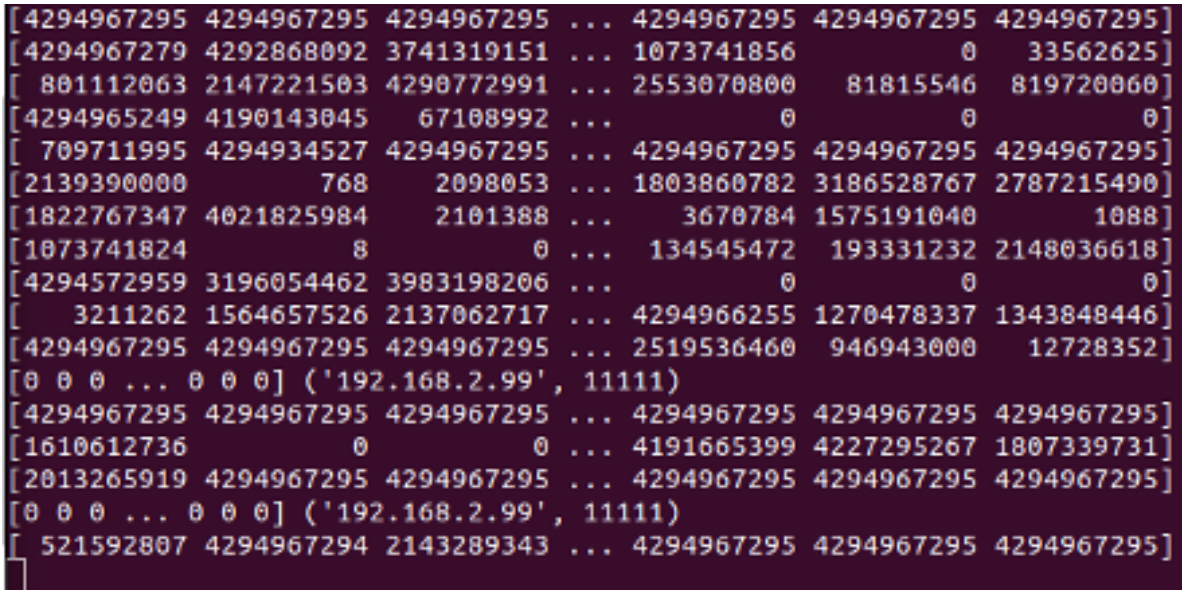


図 2.43　転送先 PC のコンソール

第 3 章

結果・将来展望

3.1 結果

本研究では、SoC-FPGA である Zynq 7000 を用いたシリコン検出器の読出回路のロジックパートの設計を完了した。開発要素に対する達成項目は以下の通りである。

センサ・ADC の制御モジュール 使用する機器の仕様書通りのタイミングで制御信号を生成するモジュールを開発した。

一時的にデータを貯蔵するバッファ FIFO 方式のバッファの読み書きロジックを設計した。具体的な深さについては、実機を用いた測定により一度に転送できる上限が設定できた。

データをメモリに転送するスキーム DMA 転送を採用し、DMA コアの規格に沿った形式で信号を接続することができた。ただし、DMA コアの制御ソフトウェアの設計・開発には至らなかった。

メモリのデータを外部 PC に転送するネットワーク Linux ライブラリを使用し、SoC-FPGA から外部 PC へとデータを転送するネットワークプログラムを設計した。

DMA 転送については PYNQ を用いて転送速度を計測し、約 300MBps の内部転送速度を得た。メモリデータ読出およびデータ転送プログラムは設計が完了しており、最大 1Gbps でのネットワーク通信が可能であることを確認した。しかしながら、Eclypse-Z7 への PYNQ の実装には至らなかったため、DMA コアのバインド・DMA コアの制御レジスタへの書き込みおよびステータスレジスタの読み込みを行う DMA 制御ソフトウェアの設計が課題として残った。

3.2 将来展望

今後の研究では、読出速度高速化にむけたいくつかの開発要素が考えられる。まずは、DMA 転送の制御ソフトウェア設計が急務である。PYNQ を Eclypse-Z7 へ実装を行って PYNQ ライブラリを利用するか、PYNQ を実装せずに独自のものを設計する選択肢がある。また、本研究ではセンサのデータに特段処理を施さずに読み出している。SoC-FPGA の能力を十分に発揮するには、1.4 に示したようなデータ処理ロジックを実装することが有効である。FIFO 書込までと FIFO 読出からのクロック周波数は 20MHz と 80MHz であるので、その差を利用すれば現時点でも十分リアルタイムデータ処理は実現可能である。また、機械学習を通して設計されたデータ選択・データ処理ロジックなどが完成すれば、それらをデータ読出フローに加えることでハードウェアアクセラレーションへの可能性が大きく期待できる。

参考文献

[1] Superkekb 加速器のルミノシティ増加の説明の模式図, 2020.

[2] CERN. Alice schematic as duaring run 3(after upgrade). Technical report, 2017.

[3] A. Devices. Ad7476 referance.

[4] D. inc. Eclypse z7 reference.

[5] D. inc. Pmod ad1 regerence.

[6] Marubun. Fpga とは？, 2022.

[7] V. Ramona. Ultrarelativistic Heavy-Ion Collisions. Elsevier Science, 2007.

[8] xilinx inc. Axi direcr memory access documentation.

[9] xilinx inc. Zynq 7000 soc.

[10] 浜松ホトニクス. s11865/11866.

[11] 浜松ホトニクス. 高エネルギー粒子用 si 検出器.

[12] 浜松ホトニクス. X 線非破壊検査用製品, 2023.

謝辞

本研究は、大変たくさんの方の支えなしには成し得ないものでした。この場をお借りして最大限の感謝を示したいと思います。指導教員である山口頼人准教授は、継続的に研究に取り組むための目標設定や研究計画、さらには結果にいたる過程の正当性の議論を非常に親身になって付き合ってくださいました。また、厳しくも研究以外の面でも悩みの相談に乗ってくださり、大変支えになりました。ありがとうございました。また、荻野雅紀研究員、長崎総合科学大学の大山健教授、浜垣秀樹特命教授、サイエナジー株式会社の竹内遥祐氏には FPGA を用いた回路設計という知識・経験共に困っていた際にはエキスパートとしてのご意見をいただきました。自分にない着想や着眼点からの意見で、質問に答えられないことが多々ありましたが、少しは力をつけられていたらいいなと思います。ありがとうございました。志垣賢太教授からは、その多岐にわたる経験から非常に生きた意見を多数いただきました。また自分の研究生活に関して、事務手続きや課外活動などで大変お世話になりました。本間謙介准教授からは、いつも自分の理論の詰めが甘い箇所や、斬新な角度からの質問をいただき、発表資料づくりの支えになりました。三好隆博助教の何気ないコメントは、いつも自分の心にくるものがありました。八野哲 WPI 特任助教は、研究室ミーティングの際に厳しくも核心をついた質問やコメントをいただき、お尻を叩いてもらいました。みなさん大変お世話になりました。

研究室のみなさんのおかげで、たいへん充実した研究室生活を送ることができました。感謝を述べたいと思います。それぞれの共通の趣味から、自分は 2 つの部活動に参加させてもらいました。みなさんと過ごした活動では、非常に幸せな時間を過ごせました。広島で過ごした時間がとても良いものになったのは皆さんのおかげです。部活動 O では、たくさんのお店に足を運び、とても美味しい料理やお酒を楽しめました。部活動 M では、とても白熱した戦いを経験させてもらいました。皆さんの成長を楽しみにしています。

そしてここまで支えてくださった家族に向けて今一度最大限の感謝を申し上げます。苦しい時、みなさんの差し伸べてくれた手の温かさは決して忘れられません。

最後になりますが、修士生活を代表して一句読まさせていただきたいと思います。

九死一生三度ありて 四面楚歌にて天女あり 災禍再びまみえれば 天下双つと無き誉かな