修　士　学　位　論　文

FPGA 搭載型アクセラレータカードによる ファームウェア開発・検証機構の構築

令和 5 年 2 月 3 日

専 攻 名　物理学専攻
学籍番号　216S120S
氏　　名　丸元　星弥

神戸大学大学院理学研究科博士課程前期課程

概要

素粒子実験は加速器で作る高エネルギー粒子の衝突反応や、宇宙から飛来する粒子を観測することで、標準模型の精密測定や標準模型を超える物理の探索を行う。特に規模の大きい実験では検出器のチャンネル数が膨大なため、必要なデータのみを選別し保存するトリガーシステムを採用している。トリガーシステムは一般的に、高速で複雑な処理が必要なため、書き換え可能な論理回路 Field Programmable Gate Array (FPGA) を用いていることが多い。FPGA でトリガー処理が行われた後、読み出されたデータはストレージサーバに蓄えられる。

近年素粒子実験は、実験精度の向上のため加速器のルミノシティ増強や検出器の大型化などが行われており、観測する粒子数や検出器の信号数も増え、データサイズが増加傾向にある。それに伴いトリガー処理で扱う信号の数も増加し、FPGA に実装される論理回路はその都度大規模化 · 複雑化している。トリガー性能を向上させて不要なデータを排除することが必要になるが、性能を向上させるためには新しいトリガーロジックを開発する必要がある。複雑なトリガーロジックの開発を行うために、その検証を効率よく行う機構が必要である。そこで、本研究では FPGA を用いたトリガーロジックのファームウェア開発・検証機構を構築した。

ファームウェア開発・検証機構の構築には、FPGA 搭載型アクセラレータカードを用いる。アクセラレータカードは PC に接続することで処理の一部をカード上のハードウェアで行うことができるデバイスである。ファームウェア開発・検証機構はトリガーロジックをアクセラレータカード上の FPGA に実装することで、トリガーロジックとトリガー性能を検証できる。本論文では、このデザインを LHC-ATLAS 実験第三期運転の初段エンドキャップミューオントリガーを例として、ファームウェア開発・検証機構の性能を評価した。

目次

第 1 章

序論

物質を構成する最小の単位は素粒子であり、標準理論は素粒子とその相互作用を記述する理論である。図 1.1 に示すように標準理論では 12 種類のフェルミオンと 4 種類のゲージボソン、ヒッグス粒子の計 17 種類の粒子が導入されている。自然界には物質と物質の間に力が働き、そうした力も素粒子が媒介すると考えられている。電荷をもつ素粒子同士に働く電磁気力は、フォトンが伝える。陽子や中性子が原子核としてまとまるのは、グルーオンが媒介して伝える強い力の働きによる。そして弱い力は二種類のウィークボソンによって媒介される。

素粒子に関するあらゆることを記述できる標準模型ではあるが、それを超える物理も存在する。標準模型を超える物理は、標準模型では説明できない強い CP 問題、ニュートリ

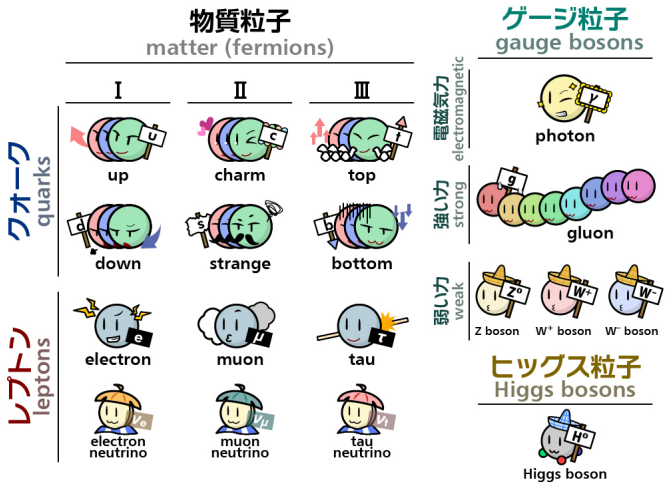


図 1.1: 標準理論を構成する素粒子の一覧 [1]。

ノ振動、物質-反物質非対称性、暗黒物質やダークエネルギーの性質などを説明するために必要な理論の拡張のことを総称する [2]。現代の素粒子物理学において標準模型をより深く理解し、それを超える物理を探求することは物理学の発展において最も重要なことである。

素粒子実験は加速器で作る高エネルギー粒子の衝突反応や、宇宙から飛来する粒子を検出器で観測することで、標準模型を構成する粒子の精密測定や標準模型を超える物理の探索を行う。世界最高輝度の電子・陽電子衝突加速器 SuperKEKB での Belle-Ⅱ実験 [3]、5 万トンもの超純水を用いて宇宙線の観測を行うスーパーカミオカンデ実験 [4]、欧州素粒子原子核機構 (CERN) のハドロン衝突型加速器 LHC [5] における世界最高エネルギーのコライダー実験があげられる。

一般的に素粒子実験のデータ取得システムでは、データ取得の必要があるものだけを保存するためのトリガーシステムを採用している。例えば LHC 加速器の衝突点の 1 つで行われている LHC-ATLAS 実験 [6] では、40 MHz の高頻度で陽子-陽子衝突が行われている。計算機資源の制約からすべての衝突事象を保存することが不可能であり、膨大な衝突事象から物理的に興味のある事象に選別するトリガーが重要な役割を担う。この衝突頻度ではトリガーはハードウェアベースで高速に処理を行う必要があり、書き換え可能デバイス Field Programmable Gate Array (FPGA) にトリガーロジックを実装している。

第 2 章で詳細を述べるが、近年の素粒子実験は加速器ビームの高輝度化や、検出器の大型化 · 精密化が進んでいる。これらのアップグレードに伴ってトリガーロジックも変更を行う必要があり、より大規模化 · 複雑化していくトリガーロジック開発の効率化が必須となる。本研究は、トリガーロジックの開発を行う上で必要なトリガー検証ができる機構を作成することが目的である。

本論文では PC に接続することで CPU の処理の一部を代替することができる FPGA 搭載型アクセラレータカードというデバイスを用いて、この課題を解決するファームウェア開発 · 検証機構の構築手法について述べる。第 2 章で素粒子物理実験のトリガーシステムとファームウェア開発の必要性について、第 3 章ではファームウェア開発・検証機構の要請とデザインついて説明する。さらに、第 4 章では、ATLAS 初段ミューオントリガーを用いて開発したファームウェア開発 · 検証機構の評価を行い、最後に第 5 章で今後の展望と本研究のまとめについて述べる。

第 2 章

素粒子実験とトリガーシステム

2.1 素粒子実験

素粒子実験は、加速器で作る高エネルギー粒子の衝突反応の観測や、宇宙から飛来する粒子を検出器で観測し、標準模型の精密測定や標準模型を超える物理の探索を行う。素粒子実験は大きく 2 つに分けることができ、加速器を用いるものとそうでないものがある。スイスのジュネーブにある円形の衝突型加速器 LHC では、重心系エネルギー 14 TeV で陽子ビーム同士を衝突させることで生じる粒子を観測する加速器実験を運行している。その衝突点の一つで行われている LHC-ATLAS 実験では、衝突反応から生じる粒子を大型検出器で観測している。2012 年にはヒッグス粒子を発見し、標準理論を構成する素粒子すべてが観測できたことに寄与した [7]。宇宙線観測の代表的な素粒子実験としては、スーパーカミオカンデがある。スーパーカミオカンデは 5 万トンの超純水を用いてニュートリノを観測している。1998 年にはニュートリノ振動を発見し、ニュートリノが質量をもつことを証明した [8]。素粒子標準理論ではニュートリノ質量は 0 だと考えられていたため、標準模型を超える物理の存在を決定づける画期的な発見となった。

2.1.1 素粒子実験のデータ取得システム

素粒子実験の一般的なデータ取得システムの概略を図 2.1 に示す。検出器に粒子が飛来した際に生成されるアナログ信号は適度な電圧に増幅 · 整形された後、デジタイザを用いてデジタル化する。実験規模が大きいものでは検出器の大きさ · 衝突頻度の多さからチャンネル数 · イベントレートが膨大なため、それらをすべて保存することはハードディスク容量による制限を考慮すると現実的ではない。そのため、電子回路で構成されたデータの

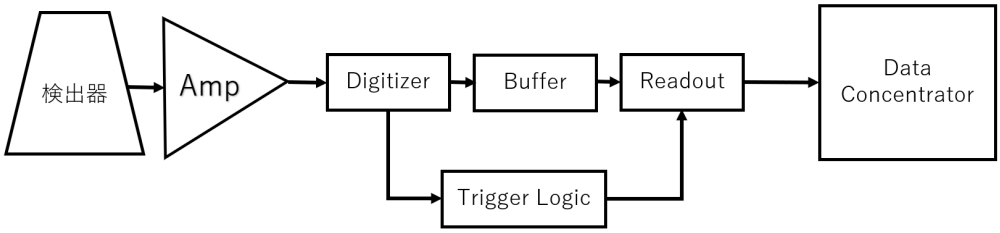


図 2.1: 検出器実験におけるデータ取得システム

取捨選択を行うトリガーロジックを通してイベントレートを削減している。これにより興味のある事象を残しながら、データ量の削減を実現する。

トリガーロジックは高速なイベント選別を実現するために、Field Programmable Gate Array (FPGA) に実装されることが多い。FPGA はその名の通り、使用者により内部ロジックの変更が可能なデジタル集積回路である。FPGA 内部は以下のような回路で構成されている。

Look-Up Table (LUT)
: すべての入力の組み合わせに対して事前定義された出力のリストを記憶し、論理操作の結果を出力する機構。

Frip-Frop (FF)
: 1-bit のデータを保持することのできる機構。レジスタに使用される。

Multiplexer(MUX)
: 2 つ以上の入力を 1 つの出力にする機構。組み合わせ回路に用いられる。

Block Random Acccess Memory (BRAM)
: FPGA 上にあるメモリ。特定のアドレスに値を保持することができ、自由に読み書きを行える。

これらの機構の配線を組み合わせることでユーザーが指定した特定の処理を行える。

現行の LHC-ATLAS 実験第三期運転では 40 MHz の高頻度で陽子バンチが衝突するが、CPU リソース・データ保存ディスク容量の制約上、データを保存できるレートは約 1 kHz (1 Gb/s) である。そのため、FPGA などを用いた高速に処理を行う初段トリガーシステムによって、2.5 μs 以内に 100 kHz までイベントレートが絞られる。初段トリガーで残った事象に対し、ソフトウェアで記述された後段トリガーによって、時間をかけてより詳細な選別を行うことで 1 kHz までレートを削減してデータを保存する。

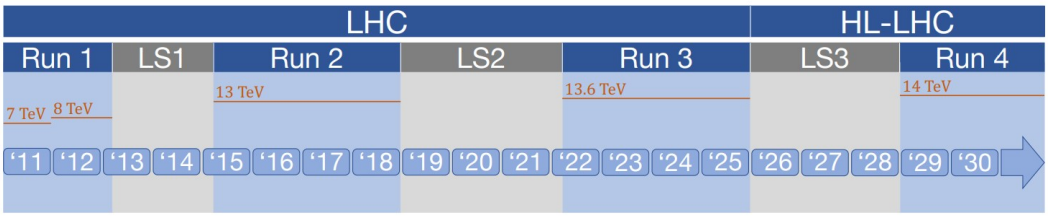


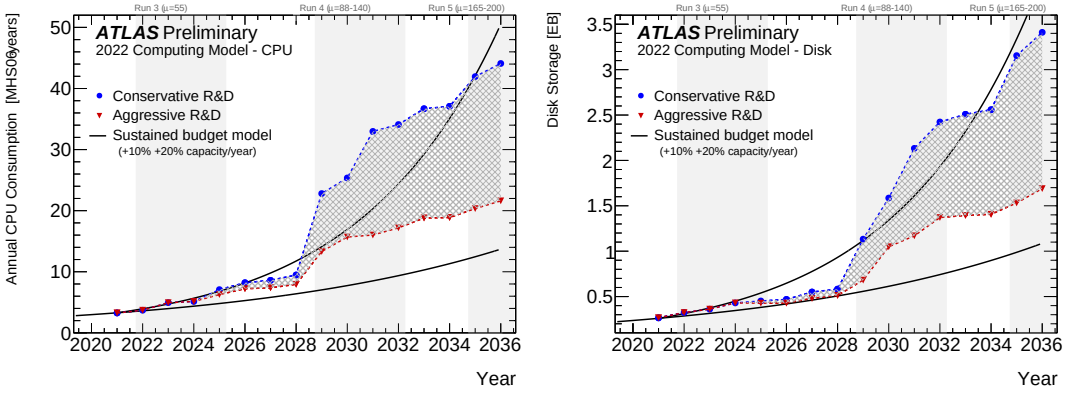
図 2.2: LHC 加速器の運行スケジュール [9]。2022 年から 2025 年まで Run 3 の運転が行われ、アップグレード期間を経て高輝度 LHC として運転が開始される予定。

2.2 ファームウェア開発・検証の必要性

素粒子実験において、標準模型を超える事象は非常に断面積が小さく、稀なイベントであることが期待される。そのため統計量を増加させることで探索精度を向上させ、新物理の発見を目指している。統計量を増加するためには事象数を増やす必要があり、そのために素粒子実験では検出器の大型化 · 精密化やビーム粒子の高輝度化 · 高エネルギー化が近年行われている。例として、日本の大型ニュートリノ実験であるスーパーカミオカンデは、2027 年頃までに約 10 倍の有効体積を持つハイパーカミオカンデへとアップグレードしデータを収集する予定であり、統計量の増加が見込まれる。さらに、スーパーカミオカンデの 2 倍も感度が高い超高感度光センサーの導入も予定されている。ビーム粒子の高輝度化 · 高エネルギー化の例として LHC では、衝突点付近のビームの絞り込み · ビーム粒子数の増加を行い、最大瞬間ルミノシティを現行の 3 倍近く増加させる、高輝度 LHC (HL-LHC) 計画がある。HL-LHC は 2029 年から 2040 年までのデータ取得期間が予定されている。LHC-ATLAS 実験の運行スケジュールは図 2.2 のようになっている。

ルミノシティ増強などによるデータ量の増加は、より多くの計算機資源量を要求する。例として、LHC-ATLAS 実験での年代ごとに必要となる計算機資源量の見積もりを図 2.3 に示す。継続的に計算機資源に投資する予算を増やしていった場合でも、HL-LHC の運転が始まる 2029 年頃から必要なリソースが急激に増加する事に対処できないことが分かる。従って、設備投資のみで対応するのはコスト面からも非現実的であり、様々なアプローチから計算機資源量の削減を行う必要がある。

計算機資源の許容範囲内で新物理の探索感度を最大化するためには、本当に取得したい事象のみを保存するトリガーシステムのアップグレードが必要である。トリガーシステムにより、データ保存量の削減及びデータ処理に用いる CPU リソースの削減を行う事がで



(a) CPU リソース (b) ディスク容量

図 2.3: ATLAS 実験において、年代ごとに必要となる計算機資源量の見積もり [10]。これからの研究開発による技術の進展を無難 (青線) に見積もった場合と、挑戦的 (赤線) に見積もった場合の、計算機使用量の推移の予測を表している。図中の実践は毎年継続的に投資を行うことを想定し、一定のコストで新しいハードウェアの能力が年率 10%、20% 向上することを示している。縦軸は HEP-SPEC06 (HS06) という指標を単位として表されている。HS06 とは、Worldwide LHC Computing Grid (WLCG) において CPU ベンチマーキングの指標として用いられている量であり、HEPiX Benchmarking Working Group [11] によって制定されている。

きる。トリガーロジックは近年、機械学習を取り入れた新しいアプローチも研究されており、短いレイテンシで高性能なロジックが期待されている [12]。これにより、トリガーロジックはより大規模化・複雑化しており、ATLAS 実験では表 2.1 に示すような大規模な FPGA が使われているのが分かる。現在のトリガー検証は複数のプラットフォームを必要としており、手間が多いことが問題となっている。将来のトリガー開発を効率化するためにも、開発したロジックの検証を行うような機構を 1 つのプラットフォーム上で実行できる環境が求められる。

そこで本研究では、これからのトリガーロジック開発の効率化を目的として、トリガーロジックの開発・動作検証・性能検証を行えるようなファームウェア開発・検証機構を構築する。開発したトリガーを 1 つのプラットフォーム上で簡単かつ高速に検証できることで効率的なトリガー開発が実現できる。

表 2.1: LHC-ATLAS 実験初段エンドキャップミューオントリガーで用いられた FPGA とリソースの推移 [13–15]。第一期 (Run-1) と第二期 (Run-2) は同じ FPGA が用いられた。2029 年から始まる HL-LHC (Run-4) では現在行われている Run-3 の 3 倍の BRAM を備えた FPGA が用いられる。

	XC2V3000 Run-1/2 (2011∼2018)	XC7K410T Run-3 (2022∼2025)	XCVU13P Run-4 (2029∼)
Logic Cells	57,344	406,720	3,780,000
SRAM	1,728 Kb	0	0
BRAM	0	25,740 Kb	95,768 Kb
URAM	0	0	368,640 Kb

第 3 章

ファームウェア開発・検証機構

第 2 章では素粒子物理実験のイベントレート増加に伴い、トリガーロジックも高性能化する必要があり、トリガーロジックの開発及び性能検証の効率化が必須であると述べた。本章ではトリガー検証のためのファームウェア開発 · 検証機構の要求について述べ、必要となるデバイス及びファームウェア開発のデザインについて説明する。

3.1 ファームウェア開発・検証機構

図 3.1 にファームウェア開発・検証機構の概略を示す。トリガー検証に用いるデータを準備し、トリガー処理を FPGA で行ったものと、ソフトウェアで正確な答えを用意したものを比較することでトリガー性能を検証できる。また、ソフトウェアツールを使用してトリガーロジックの開発も行えるようにする。

3.1.1 ファームウェア—開発・検証機構の要求

トリガー開発・検証のためのファームウェア開発 · 検証機構には様々な要求がある。そのユースケース図 3.2 を示す。以下でその具体的な要求について述べる。

イベントシミュレーションが走る

トリガーの検証が効率的に行えるにはシミュレーションイベントの作成も同時に行いたい。シミュレーションイベント作成をトリガー検証と同時に行えれば、膨大なデータを用いた性能検証が可能になる。そのためには、シミュレーションソフトウェアが走る CentOS を用いる必要がある。本デザインではこの要求を満たすようなデザインを行う。

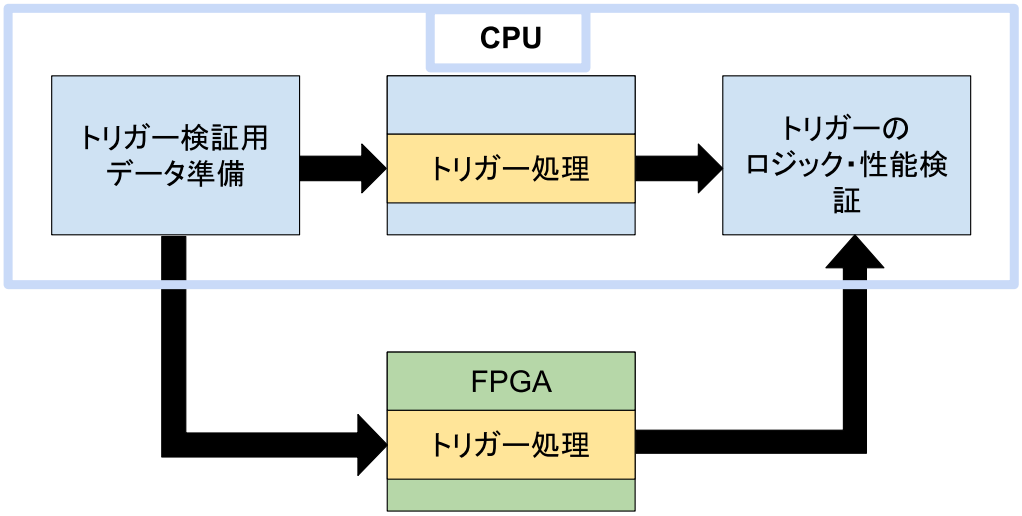


図 3.1: デザインするファームウェア開発 · 検証機構の概略。青い領域はソフトウェア処理、緑の領域は FPGA で処理される部分である。検証データを用意して、FPGA でトリガー処理を行った結果をソフトウェア処理結果と比較して検証を行う。

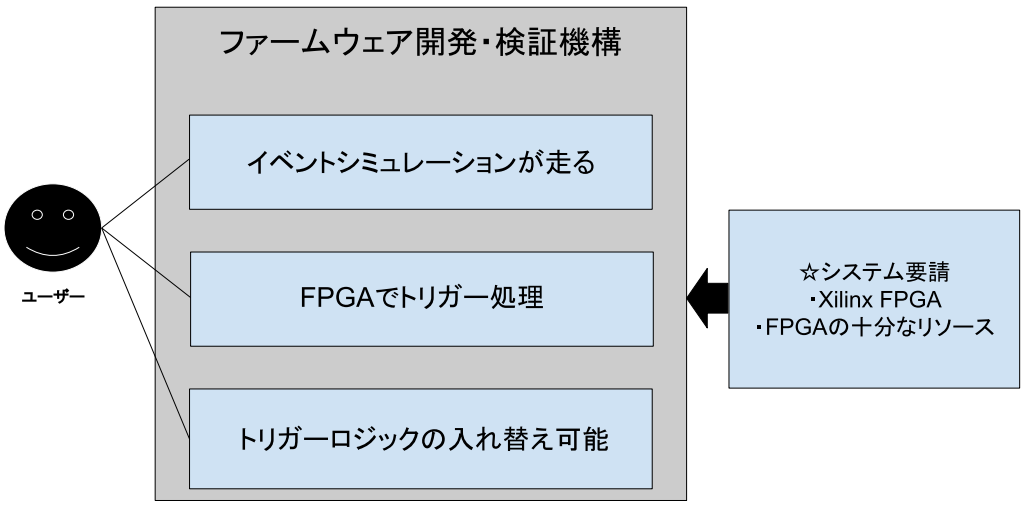


図 3.2: ファームウェア開発・検証機構の要請

FPGA にトリガーロジックを実装できる

トリガーロジックの検証は FPGA で行うことが必要である。実験で用いられているトリガーロジックを本デザインに組み込むことでトリガー処理部分を FPGA 上で行えるようなデザインを行う。

トリガーロジックの入れ替えが可能

これまでのトリガー検証は検出器の実機上でトリガーの検証をしていた。そのため、実装するまで検証を行うことができないことや、実機上のボードに入れるデータ入力に膨大な時間がかかることから、特定のイベントに対するトリガー検証しか行えていなかった。本検証機構ではトリガーロジックの入れ替えができ、FPGA と高レートでデータ転送を行うことで様々なトリガー検証が容易に行えるようにする。

これらの要求を満たすうえでさらに以下のようなシステム要件を設定した。

- 素粒子実験で多くに用いられている AMD Xilinx 製の FPGA を用いること。
- トリガーロジックを搭載するための十分な FPGA のリソース量が必要。第三期運転に用いられた FPGA では BRAM が搭載されており、トリガーに使用されている。2029 年から行われる運行では 2025 年の約 3.5 倍近くのリソースに増える。本デザインでは BRAM の搭載されている FPGA であり、少なくとも XC7K410T よりリソースが多いことを要求する。
- 様々なイベントに対して検証を行えるような演算能力。ATLAS のシミュレーションイベント作成ソフトウェア Athena [16] が駆動する CentOS を搭載した PC を用いること。
- 可能な限り安価に購入ができること。

本研究ではこれらの要求を満たす FPGA 搭載型アクセラレータカードである AMD Xilinx 社の Alveo アクセラレータカード [17] を用いることにした。

図 3.3: Alveo U200 アクセラレータカード [18]。

表 3.1: Alveo シリーズの FPGA と主なロジックエレメントにおけるリソース [19]。

	Alveo U200	Alveo U250	Alveo U280	Alveo U50
FPGA	XCU200	XCU250	XCU280	XCU50
BRAM	77,760 Kb	95,768 Kb	72,576 Kb	48,384 Kb
URAM	276,480 Kb	368,640 Kb	276,480 Kb	184,320 Kb
LUT	1,182k	1,728k	1,304k	872k
FF	2,364k	3,456k	2,670k	1,743k

3.1.2 Alveo アクセラレータカード

アクセラレータカードは、パソコンの処理速度を上げるための役割を果たすハードウェアのことである。ホスト PC での計算処理の一部をデバイスで行うことでアプリケーションを高速化することができる。本研究ではトリガーロジックを検証するために、Xilinx 社が開発した FPGA が載ったアクセラレータカード、Alveo データセンターアクセラレータカード U200 を用いて開発を行った。図 3.3 に Alveo データセンターアクセラレータカード (U200) のイメージを示す。

Alveo データセンターアクセラレータカードは Peripheral Component Interconnect-

表 3.2: U200 の SLR ごとに使用可能なリソース [21]。BRAM は 36Kb サイズ単位で個数を表しており、URAM は 280Kb サイズ単位での個数を表示している。

Resource	SLR0	SLR1	SLR2
LUT	388K	205K	770K
register	776K	410K	770K
BRAM	720	420	720
URAM	320	160	320

Express (PCIe) [20] インターフェースを利用することでホストとアクセラレータカード間の通信、およびホストから Alveo カードのデバイスメモリへのデータ転送に使用できる。U200 に搭載されている Gen3x16 では約 126 Gbps でデータ転送が可能である。Alveo カード上の DDR はグローバルメモリとして機能し、ホスト及び、Dynamic Region 両方からアクセスできる。ホストから直接 Dynamic Region にアクセスすることができないため、データは DDR を介してやり取りする。DDR は 4 つあり、それぞれ 16 GB のメモリを備えている。Static Region にはダイレクトメモリアクセス (DMA) があり、ホストメモリとカード上のメモリで直接データのやり取りが可能になっている。

Alveo アクセラレータカードは U50 から U200 および U250 まで様々な種類がある。表 3.1 に示すように、それぞれが異なる FPGA リソース、メモリ容量、メモリタイプを提供する。本研究では価格とリソースの観点から U200 を利用している。

Alveo のブロックデザインを図 3.4 に示す。Alveo U200 カードには Ultrascale+ XCU200 が搭載されており、複数の SLR (Super Logic Region) を組み合わせることにより集積度を挙げている。それぞれの SLR が特定の DDR メモリとやり取りすることができる。各 SLR ごとに使用可能なリソース容量を表 3.2 に示す。これは Static Region で使用できるリソースを差し引いたリソースを表している。Static Region はクロックの生成やホスト CPU との通信などに使用される領域である。各 SLR は指定された DDR メモリリソースと接続が可能になっている。

本研究で行うトリガー検証はこの Dynamic Region に実装することができる。具体的な信号の流れを説明する。まず、ホストから必要な処理を PCIe を介して Alveo カードにデータを転送する。転送したデータを DDR メモリに書き込んで置き、Advanced eXtensible Interface 4 (AXI4 [22]) 通信を介して Dynamic Region で信号処理し、結果を DDR メモリに書き込む。その後、PCIe を通してホストに DDR メモリのデータを転

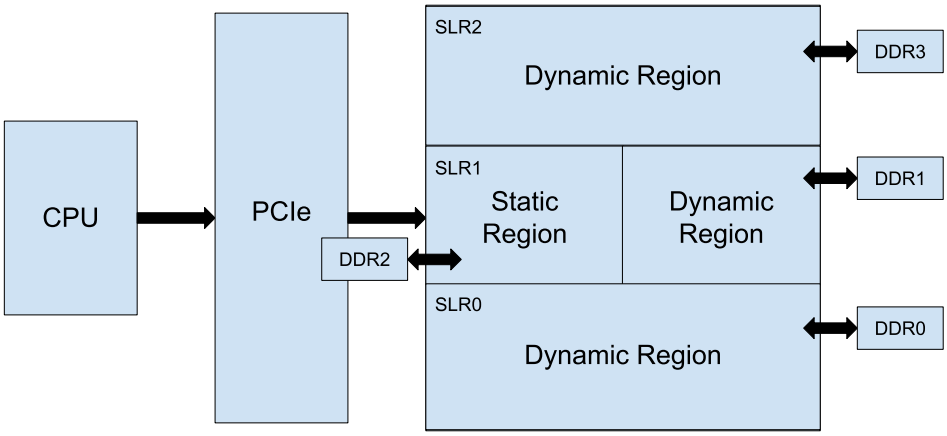


図 3.4: U200 のブロックダイアグラム。FPGA 内部の領域を SLR0,SLR1,SLR2 の三つの領域に分け、それぞれの領域が特定の DDR メモリとつながっている。CPU から PCIe を通して、Static Region における DMA を介して DDR に値を転送できる。Dynamic Region はユーザーが自由に書き換えることができる領域である。

送することで結果が得られる。

3.1.3 開発に使用するホスト PC

開発に使用する PC は図 3.5 に示したようなもので、動作周波数が 3.5 GHz の Intel 製 CPU (Core i9 10920X) を搭載で OS は Cent OS 7.9 である。この PC に Alveo U200 アクセラレータカードを搭載している。

3.1.4 開発フロー

開発は 3.1.2 節で概説した Alveo データセンターアクセラレータカード U200 を用いたアクセラレーション開発フローに従った。開発を行うにあたって用いたプラットフォームとソフトウェア API を以下に示す。

- 開発プラットフォーム xilinx_200_gen3x16_xdma_base_2
  U200 の基礎となるブロックデザインを構築するために使用する。
- Dynamic Region の開発ソフトウェア　Vivado 2021.2
  U200 の Dynamic Region をハードウェア記述言語で開発するためのツール。

図 3.5: 開発に使用した PC の写真。Alveo U200 を搭載しており、PCIe で接続している。

- ソフトウェア開発ツール　Vitis 2021.2
  C 言語などを用いたアクセラレーションアプリケーションの開発を可能にするソフトウェア開発ツール
- ソフトウェアライブラリ　XRT 2021.2
  Vitis の主要コンポーネントで、アクセラレーション開発の際に FPGA とホスト CPU のやり取りを簡単に行えるようにする API。

開発全体のイメージ図を図 3.6 に示す。開発する部分は大きく 2 つあり、CPU 側のソフトウェア開発と FPGA 側の FPGA コアの開発である。FPGA コアは Dynamic Region と呼ばれる領域のことで、論理回路の配線がつながれていないため、配線をつなぐ必要が

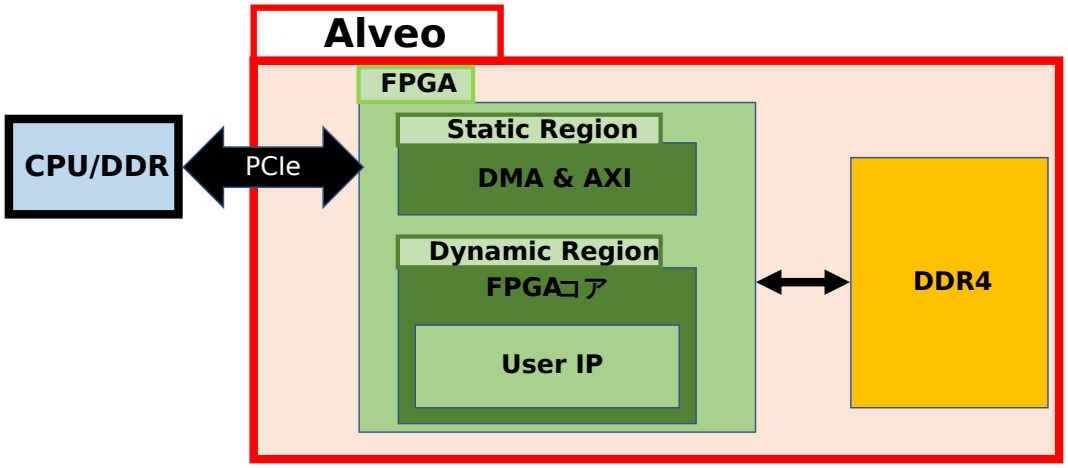


図 3.6: 開発フロー全体のイメージ。CPU 部分でソフトウェアアプリケーションの実行を行い、FPGA のうち、FPGA コア (Dynamic Region) の部分を開発してトリガーを実装できるようにした。開発はこの 2 段階の手順で行う。また、Static Region 領域は DMA や AXI などを使用して Alveo 上のメモリとホスト上のメモリ間におけるデータやり取りを実行できるようにする。

ある。FPGA コアの配線をデバイスに書き込んだ後に、ソフトウェアアプリケーションを実装して全体の処理を行う。

FPGA コアの開発は図 3.7 に示すように、C、C++、OpenCL および、ハードウェア記述言語である Resister Transfer Level (RTL) のいずれかで記述して行う。Vitis を使用することで Open CL や C/C++ で記述したロジックを用いた開発を行うことができる。また、Vivado HLS を用いても同様のことが可能で、Vivado HLS は hls4ml を用いた機械学習モデルの変換も可能である。一般的な検出器実験では詳細な設計を行うためにトリガーロジックはハードウェア記述で開発が行われているので、本研究では RTL を用いた実装を行った。3.1.2 節で前述したとおり、FPGA コアは AXI4 規格の通信で統一されており、FPGA コア開発はこの転送サイズの設定やメモリとのやり取りを決める。RTL カーネルでの記述は Vivado ツールを用いてデバイスに実装ができる。作成した RTL ファイルは Vivado IP パッケージャコマンド (package_xo) によりリンクに使用し Xilinx Object (xo) ファイルを生成する。

コンパイル後は Vitis 統合開発ソフトウェアにより、xo ファイルがハードウェアプラットフォームとリンクし、デバイス実行ファイルであるザイリンクスバイナリ (.xclbin) ファイルが生成する。

アプリケーションを含めたファームウェア開発 · 検証機構の全体実装フローを図 3.8 に

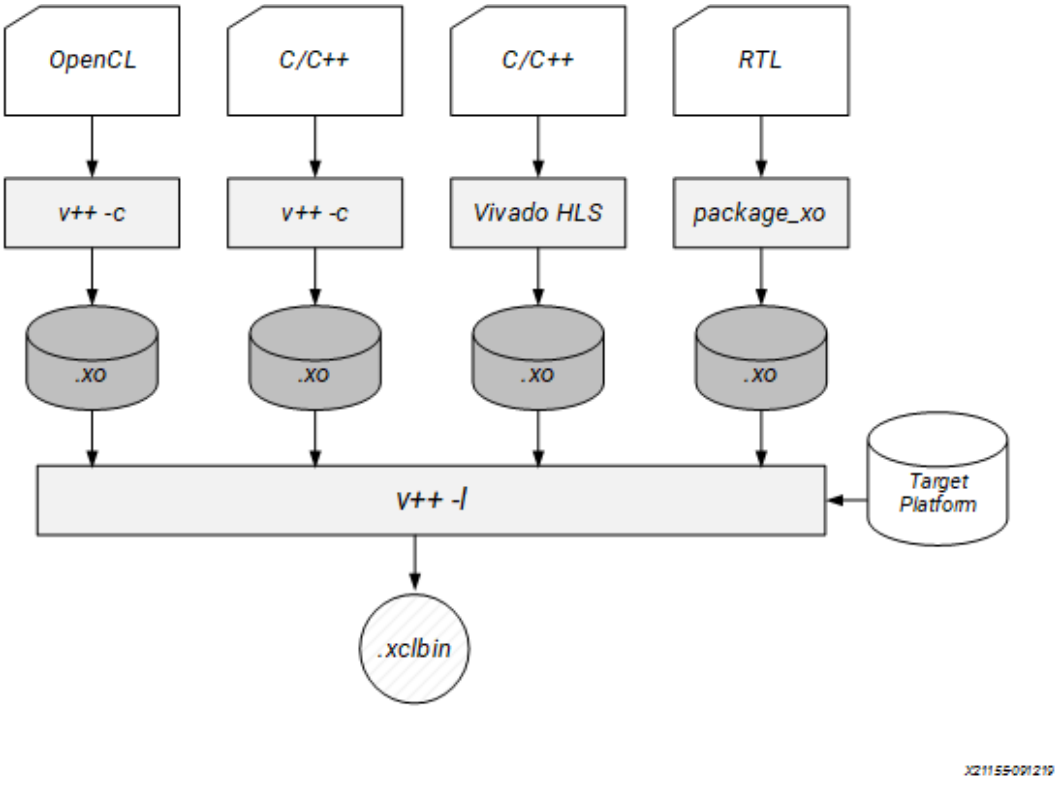


図 3.7: FPGA コアの開発プロセス [23]。C、C++、OpenCL および、ハードウェア記述言語である Resister Transfer Level (RTL) のいずれかで記述した FPGA コアの配線を Alveo で実行できるファイル形式にする。

示す。ソフトウェアアプリケーションは Xilinx Run Time (XRT) ネイティブ C++ API を使用して記述する。ソフトウェアアプリケーションではアクセラレータデバイスの指定、`.xclbin` ファイルをアクセラレータデバイスに書き込み、カーネルとカーネル引数の指定、ホスト PC·Alveo 間のデータ転送、FPGA 処理の実行と結果の読み取りを行う。

FPGA 上でのデータやり取りは AXI4 規格を使用して行われている。AXI4 は ARM 社が提供する 4 代目の AMBA インターフェース規格で、本研究で開発する FPGA コアの通信規格であり、この 1 つの規格を理解するだけで、FPGA 開発が行えるようになっている。Xilinx 社の FPGA では AXI4 の単一規格のインターフェースを採用しているため、IP の統合がより簡単になる。ザイリンクスはエンデベッド、DSP、ロジックドメインにおいて、単一のオープン規格インターフェースを採用する幅広い AXI4 ベース IP を提供している。AXI の基本的な通信プロトコルは valid 信号と ready 信号のハンドシェイクにより、data 信号の送受信を制御することである。典型的な通信の様子を図 3.9 に示

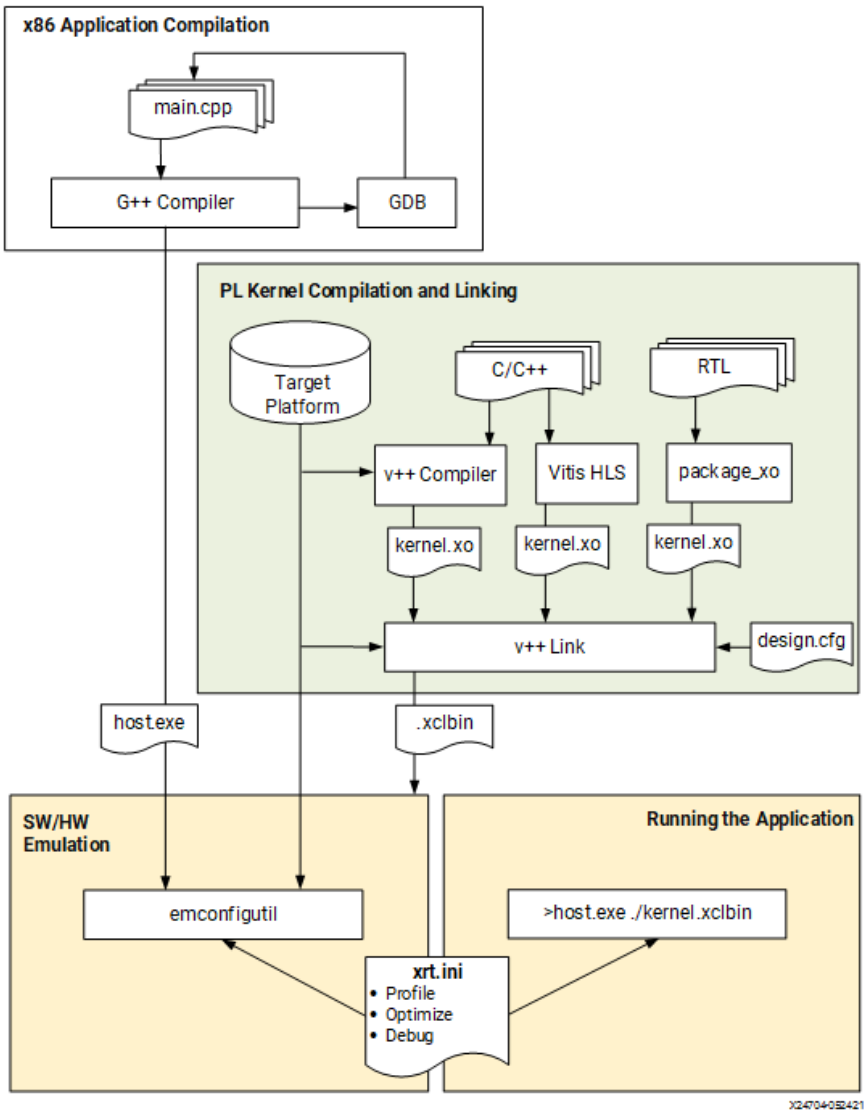


図 3.8: alveo データセンターアクセラレータカードを用いたファームウェア開発フロー [23]。FPGA コアの開発プロセスで作成したデバイス用に書かれた実行ファイル (`.xclbin`) と、ソフトウェアアプリケーション (`main.cpp`) を用いてアプリケーションの実行を行う。

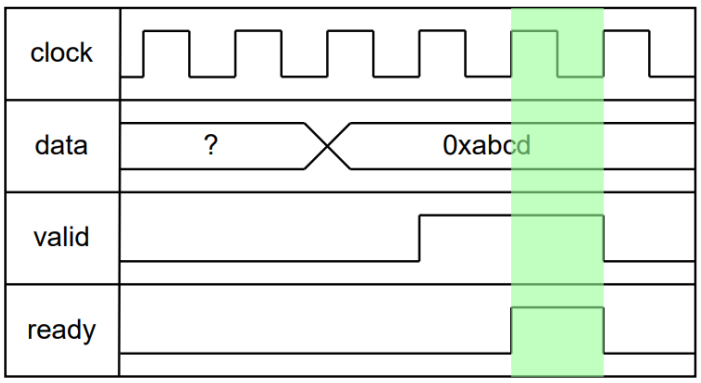


図 3.9: AXI 規格の典型的な通信の様子。valid 信号と ready 信号がともに 1 になるとデータの転送が許可される [24]。

す。valid 信号と ready 信号がともに 1 のときに通信が行われる。

AXI4 には用途に合わせて 3 つのプロトコル、AXI4、AXI4-Lite、AXI4-Stream が用意されており、どの規格もマスターとスレーブをつなぐ 1 対 1 のバスになっている。AXI ではデータ転送の要求をするのは常にマスターだが、実際のデータはマスターからスレーブへ送られることも、スレーブからマスターへ送られることもある。

本研究では AXI4 (full) 規格を使用してファームウェア開発・検証機構をデザインした。AXI4 (full) はメモリマップ式のバースト可能インターフェイスで最大 256 バーストでのデータ転送が可能である。データ幅は 8-bit, 16-bit, 32-bit, 64-bit, 128-bit, 256-bit, 512-bit, 1024-bit, 2048-bit から選択可能である。本研究では 512-bit を採用した。

書き込みはマスターからアドレスとデータを送り、読み取りはマスターからアドレスを送り、arvalid と arready の信号が同時に立ち上がっているタイミングでスレーブから送られているデータを読み取る。AXI4-Lite インターフェイスはバースト転送ができないという点以外は AXI4 (full) と同じである。

3.2 ファームウェア開発・検証機構の実装

3.2.1 実装したデザイン

作成したファームウェア開発 · 検証機構の FPGA コアを図 3.10 に示す。本論文において、FPGA コアは Dynamic Region において DDR とのデータ転送を指定した領域を示す。Dynamic Region は、U200 の書き換え可能な論理回路として実装できる部分であり、DDR とのデータやり取りを決める。動作クロックは 300 MHz で DDR からの読み出しはクロックごとに 512-bit で行われる。FPGA コアの中にトリガーロジックと置き換える部分 (User IP) を配置し、様々なトリガーを検証することができるようにした。DDR とホスト間のデータやり取りはホストアプリケーションで制御する。3.1 節でも説明したように信号の流れは AXI4 インターフェイスをベースとして実行できる。制御信号として設定した DDR メモリのアドレスに start 信号を書き込むことで AXI インターフェイスを介して DDR メモリに書き込んだデータを読み出して計算を行うことができる。作成したコアにおける具体的な信号の流れを説明する。

1. DDR の指定したアドレスと AXI4-Lite-Master の Valid 信号をつなげている。DDR の入力と出力のアドレス情報を AXI4-Master に送る。
2. 情報を受け取った AXI4 が DDR の受け取った入力アドレスから順番に値を 512-bit ずつ読み出していく。
3. 読み出した値を User IP に送り、実装した User IP の処理を行い、出力に渡すための結果を得る。
4. User IP の結果を DDR メモリの出力のアドレスから順番に書き込んでいく。
5. 書き込みが完了すると、AXI4-Master から AXI4-Lite-Slave に実行終了を知らせる信号を送り、DDR の制御信号に User IP 処理の終了を知らせる値を書き込む。

次に、作成したファームウェア開発 · 検証機構のソフトウェアアプリケーションのデザインを図 3.11 に示す。Alveo 上の DDR に書き込むデータはホスト PC 上のメモリから DMA で書き込み、FPGA コアの制御信号に使用するデータの書き込みは CPU から Alveo 上の DDR に直接転送する。ホストコードの記述は Xilinx native API [25] を用いて行っている。一般的な実行手順を図 3.12 に示す。

1. アクセラレータデバイス ID を指定して開発したファームウェア情報 (xclbin) を Alveo に書き込む。基本的に実行時間が一番長くなる部分である。

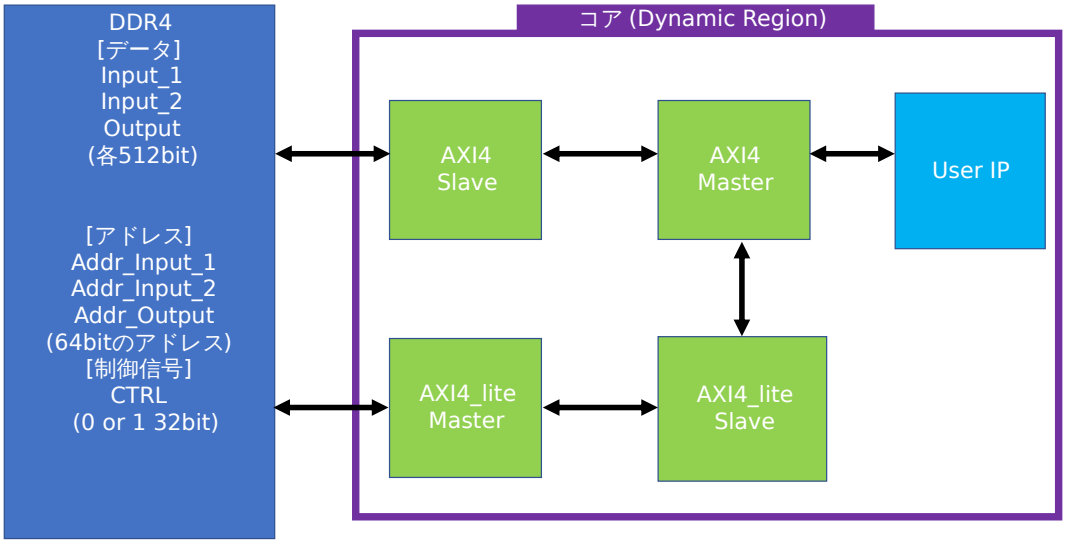


図 3.10: ファームウェア開発・検証機構のコア部分のデザイン。AXI インターフェースを用いて DDR とのデータやり取りを行い、User IP に実装した処理を行えるように開発した。AXI Lite インターフェースは FPGA コアの制御に、AXI4 インターフェースは FPGA コアと DDR のデータ転送に用いる。

2. ホストメモリに入力値を書き込む。準備する入力値は data_width 512-bit の変数２つであり、それぞれ 256 MB。
3. ホストメモリに入力値を書き込んだ後は PCIe を通して DMA によってデバイスの DDR4 メモリに値を転送する。
4. Xilinx native API では FPGA コア実行の制御をホストコードによって行う。デバイスの制御レジスタのアドレスを指定して 32-bit で値 1 のイネーブル信号を送ることで FPGA コアの実行を開始する。
5. User IP の出力結果をホストメモリに転送することで実行結果を得られる。

3.2.2 デザインしたファームウェア開発 · 検証機構のリソース

ファームウェア開発 · 検証機構のリソース使用率を表 3.3 に示す。現時点では、User IP は input と output をつないだだけで複雑な論理回路を実装していないため、使用割合が低くなっている。その中でも LUT, FF, BRAM の使用が少し上がっているのは AXI4 IP の生成に使用されるためである。

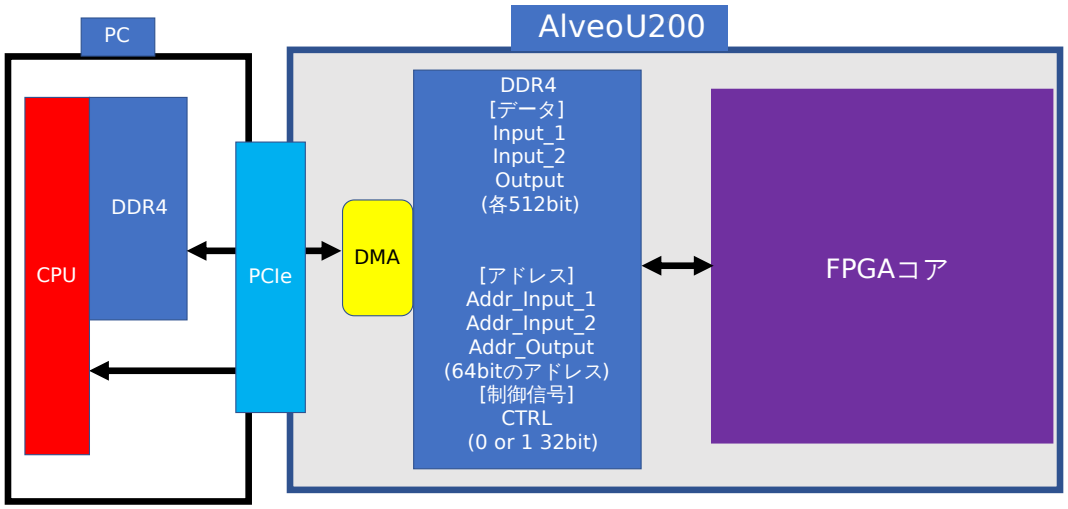


図 3.11: ファームウェア開発・検証機構のソフトウェアアプリケーションのデザイン。PC とデバイス上のデータ転送を制御したり、FPGA コアの実行を制御することでアプリケーションを構築。

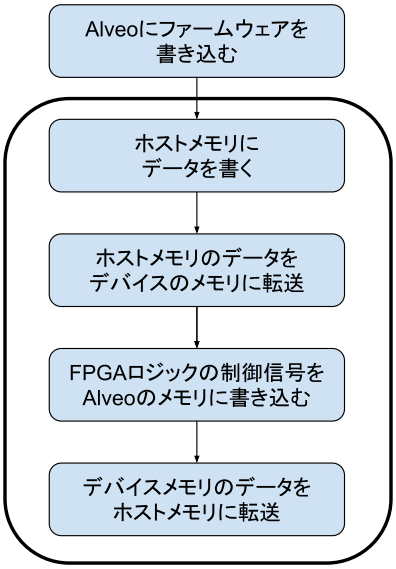


図 3.12: xilinx native API [25] を用いたホストコードの実行手順

表 3.3: FPGA コアと入力と出力の配線をつなぐ User IP のリソース使用率。AXI によるデータ転送ロジックの実装にリソースを使用している。

リソース	使用	使用可能なリソース
LUT	169,419	1,182,240
LUTRAM	15,320	591,840
FF	240,493	2,364,480
BRAM	261.5	2,160

3.3 ファームウェア検証機構の動作確認

ファームウェア開発・検証機構の動作確認のため、User IP にトリガーロジックの LUT に使用される Block Random Access Memory (BRAM) を実装した。BRAM は 5 つの入力と 1 つの出力で構成されており、addra でアドレスを入力することで dina に入力値を書き込むことができる。ena は 1-bit で書き込みと読み出しの制御を行う入力で、基本は 1 にしておくことで許可しておく。書き込みを行うときは wea に 1-bit で 1 を入力することで書き込みが許可される。逆に、wea に 1-bit で 0 を入力した場合は書き込みができないようになっている。値を出力したい場合は addra で必要な情報を入力することでそのアドレスに書き込まれた値を得ることができる。

3.3.1 データ入出力の確立と演算時間の測定

今回は、Xilinx の Vivado で用意されている Block Memory Generator を使用して BRAM を自動生成し、User IP に実装した。使用する BRAM は Dual port タイプの BRAM で 1 つの BRAM に対して、入力と出力を 2 つずつ用意することができる。Alveo U200 にある BRAM は 1 つ当たり 36 Kb のメモリアレーを有しており、プラットフォーム上では 1860 個の BRAM を使用することができる。また、1 つの BRAM は 2 つの 18 Kb メモリアレーとして使用することができる。

実装した User IP のファームウェアを図 3.13 に示す。BRAM に書き込みを行った後、実際に値が書き込めたかを出力値を見て確認する。BRAM のアドレス幅は 14-bit に設定している。アドレスは 0 から順番に増えていく単純なものとし、BRAM の入力値は 5-bit で 1 から 15 の値をループする単純なものとした。ホストからの入力値は 20-bit で 8192

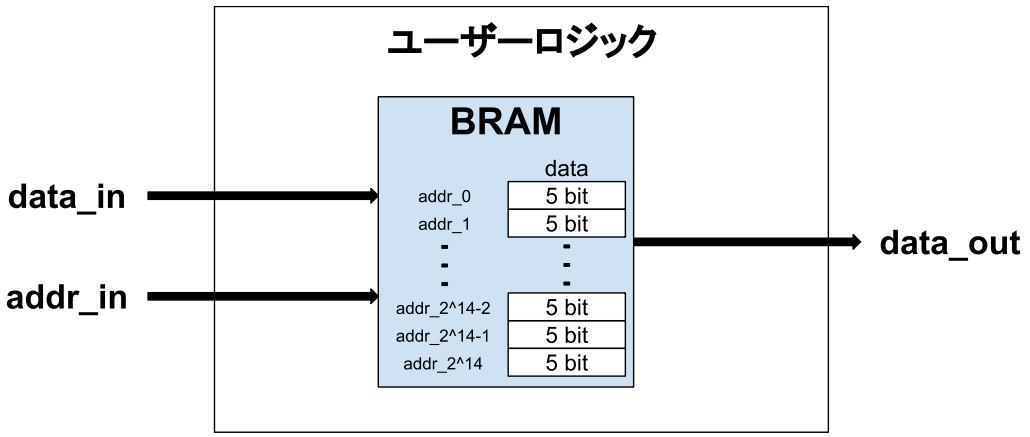


図 3.13: User IP に実装したファームウェア

表 3.4: BRAM にアクセスするための入力フォーマット。下位 5-bit は BRAM が保持する値であり、次の 14-bit に BRAM を参照するアドレスを指定する。19 bit 目は書き込みを行うかを制御するフラグ。20 bit 目は読み取りを行うかを制御するフラグに使用する。

bit	
0-4	BRAM のレジスタ幅
5-18	BRAM のアドレス幅
19	書き込みのイネーブル値
20	読み取りのイネーブル値

個用意した。入力フォーマットを表 3.4 に示す。下位の 5-bit は BRAM の保持する値。次の 14-bit が BRAM のアドレスである。次の 19 bit 目は書き込みを行うかを制御するイネーブル値。20 bit 目は読み取りを行うかを制御するイネーブル値に使用する。

作成したホストアプリケーションを図 3.14 に示す。最初にファームウェアを Alveo に実装する。ファームウェアの実装は表 3.5 でも示したように、最も時間のかかる手順であり、一度書き込みしておくことでこの時間を省略できる。青い線で囲まれた領域では BRAM へのデータ書き込みを行う。User IP の BRAM には参照値を入れていないため、値の書き込みを行わなければならない。手順としてはまず、ホストメモリと Alveo 上のメモリにバッファ割り当てを行う。ホストメモリには表 3.4 に示したファーマットでデータをメモリに書き込む。ホストメモリにデータを準備したので、DMA によりホストメモリから Alveo 上のメモリにデータを転送する。データが準備できたら、FPGA コアの実行

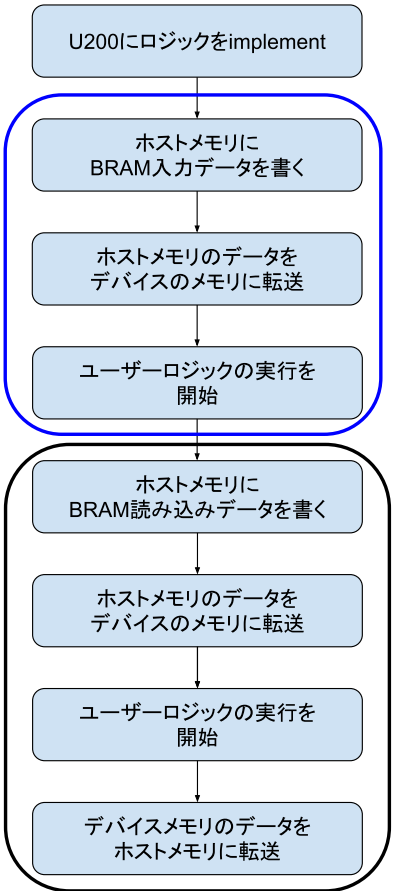


図 3.14: BRAM への書き込みと読み込みを行うアプリケーションの概略。最初にデバイスに実行ファイルをロードする。青い線で囲まれた領域は BRAM に書き込みを行うアプリケーション部分、黒い線で囲まれた領域は BRAM に書き込んだ値を読み込むためのアプリケーションで、今回は確認のために用いている (検証の際は省略)。

をトリガーすることで BRAM に値の書き込みを実行する。黒い線で囲まれた手順では前の手順で BRAM に書き込んだ値を出力して受け取るため、BRAM のアドレスを指定する入力値を用意して FPGA コアの実行をおこない、書き込み値を取得している。ここでは、読み取りを行うため、入力は $\mathrm{input}[20:19] = 01$ にすることで書き込みをディスエーブルにし読み取りをイネーブルにし、値を取得している。BRAM の入力と BRAM から読み取った値の差を取ったヒストグラムを図 3.15 に示す。入力と出力の結果がすべて正しく取れており、FPGA コアが User IP と DDR に対して適切な処理ができていることが確認できた。。

表 3.5 に xclbin ファイルの書き込みを含めたアプリケーションの実行時間をまとめた。

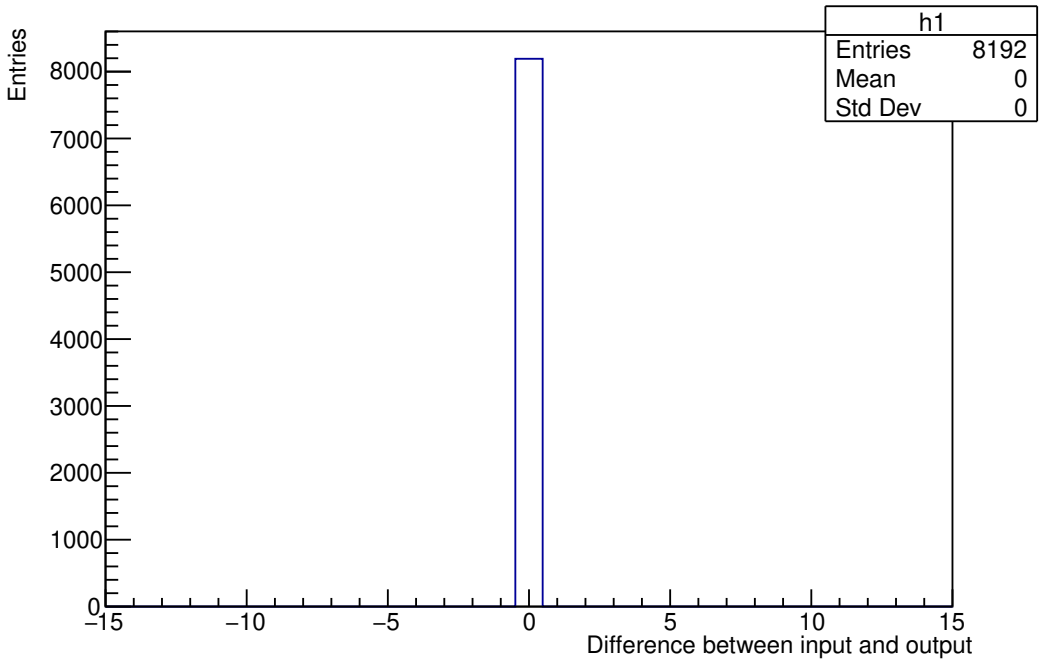


図 3.15: BRAM に書き込んだ参照値とそのアドレスにおける値を BRAM から読み込んだ値の差。BRAM のアドレスに対した参照値の書き込みがすべて正しく行えており、ファームウェア開発・検証機構の FPGA コアが正常に動作していることが確認できる。

表 3.5: Alveo にファームウェアを書き込む処理の時間。

項目	実行時間
Alveo へのファームウェア実装	2,700 ms
その他	219 ms
合計時間	2,919 ms

Alveo デバイスへのファームウェア書き込みアプリケーションの実行時間は 2919 ms である。その他で表した時間は PCIe で接続したアクセラレータカードの呼び出し時間が大半を占めている。デバイスに実行ファイルをロードする時間が約 2.7 s となっている。表 3.6 にファームウェアを書き込んだ状態でのアプリケーション実行時間をまとめた。Alveo にファームウェアを書き込んでいるので実行時間は大幅に削減されている。メモリに割り当てた値のサイズは 256 MB (DMA の最大転送レート) で設定。

表 3.6: アプリケーションの実行時間。

項目	実行時間
Alveo からのファームウェア情報読み取り	17 ms
host と Alveo メモリのバッファ割り当て	155 ms
Alveo への DMA 転送	96 ms
FPGA コアの処理時間	228 ms
Alveo からの DMA 転送	80 ms
その他	1,401 ms
合計時間	1,977 ms

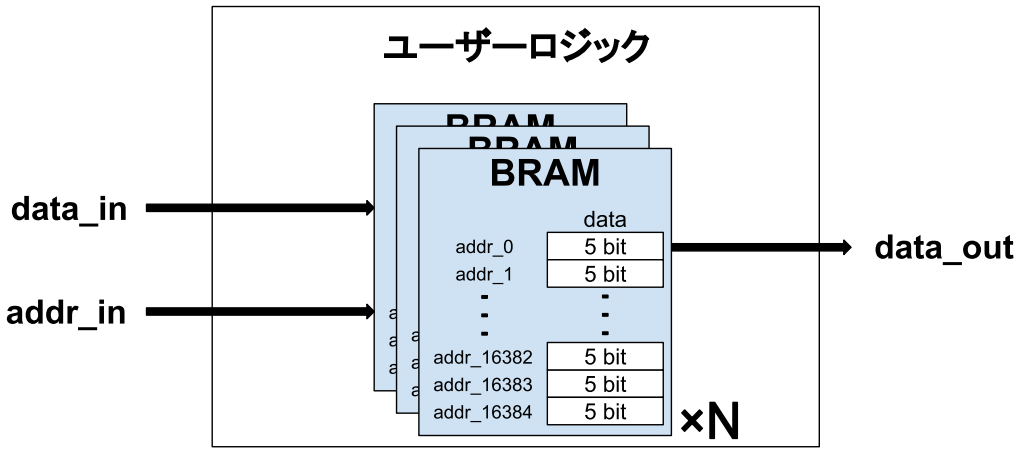


図 3.16: 搭載する BRAM の数を変化させるときの User IP のファームウェア概略。

3.3.2 BRAM とリソースの対応関係

トリガー回路に用いられることが多い BRAM の搭載数とメモリサイズを変化させ、リソース使用率の変化量を評価する。メモリサイズの変化はレジスタ幅とアドレス幅の 2 通りで調べる必要がある。最初に BRAM の搭載数の変化に対するリソースの評価を行う。BRAM は LUT でよく使用されるメモリであり、その許容搭載数を調べる。User IP には図 3.16 に示すように、単に BRAM の数を変えて実装を行った。

User IP に搭載した BRAM はレジスタ幅 5-bit、アドレス幅 14-bit で 80 Kb のサイズ

表 3.7: 14-bit のアドレス幅にレジスタ幅を 5-bit とした BRAM を 1,2 つ搭載したときのリソース使用量。使用量 (コアのみ) は User IP に何も搭載していないときのリソース使用量。使用量 1 は何も搭載していないときと BRAM1 つ搭載したときの比較。使用量 2 は BRAM の搭載数が 1 つのときと 2 つのときの比較を表している。

リソース	使用量 (コアのみ)	使用量 1		使用量 2	
LUT	169,419	169,389	(-30)	169,406	(+17)
LUTRAM	15,320	15,303	(-17)	15,305	(+2)
FF	240,295	240,394	(-99)	240,403	(+9)
BRAM	261.5	264	(+2.5)	266.5	(+2.5)

で行った。表 3.7 は BRAM を搭載していないとき (User IP の入出力をつないだだけの場合) と、前述したサイズの BRAM を 1 つ/2 つ搭載した際のリソースの比較。使用量 1 から、入力と出力のワイヤーを接続した User IP に比べて BRAM 以外のリソース使用が減っていることが分かる。これは BRAM を搭載したことにより、入力と出力のワイヤーをつなぐ論理回路が減ったためである。また、BRAM は 2.5 個消費されており、18 Kb のブロック 5 つ使われていることが分かる。使用量 2 も同様に BRAM の値が 2.5 個増えていることが確認できる。

図 3.17 に搭載した BRAM の数と BRAM リソースの使用率の関係を表した。BRAM の搭載数を 1 つ増やすごとに、2.5 個の BRAM が消費される。BRAM は 18 Kb 単位で使用されるので、2.5 個では 90 Kb 使用されるが搭載した BRAM は 80 Kb サイズであるので、10 Kb 余分に使ってしまっている。User IP に搭載していない状態では AXI4 によって 262 個の BRAM が消費されていたので、最大搭載可能数は 1598 個であり、レジスタ幅 5-bit、アドレス幅が 2^{14} 通りのデュアルポート BRAM は最大で 639 個搭載できることが分かる。

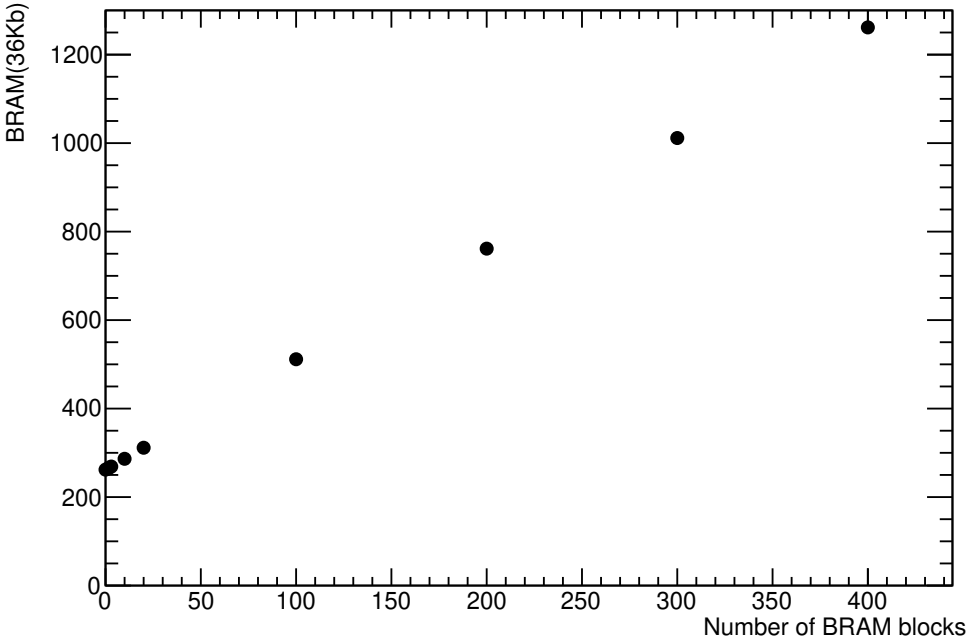


図 3.17: 書き込みデータサイズ 5-bit、アドレス幅 2^{14} 通りの BRAM を搭載した数とそのリソース使用数の関係。

次に BRAM のメモリサイズの変化に対するリソースの評価を行う。BRAM のリソース変化はアドレス幅とアドレスごとに入力できるレジスタ幅の 2 通りある。まず、BRAM のレジスタ幅を変化させ、それに伴ったリソースの変化量を評価する。User IP のファームウェアは図 3.18 に示すように、BRAM のレジスタ幅のビット数 n の値を変えることで、そのリソースの変化を見積もる。アドレス幅は 14-bit に固定した。Xilinx IP の BRAM のレジスタ幅は 1-bit から、最大 4608-bit まで可能である。

レジスタ幅が 10-bit の時のリソース使用率を表 3.8 に示す。使用変化量は表 3.7 と比較したものでレジスタ幅が 5-bit の場合と比べている。bit 数が倍になることで BRAM 消費数の変化は 2 個となった。レジスタ幅 5-bit、アドレス幅 14-bit の BRAM の数を 1 つから 2 つに増やしたときは、BRAM の使用数が 2.5 個上がったのに対して、消費量が削減されている。どちらの場合も 80 Kb サイズの BRAM が増えているが、レジスタ幅を増やした場合は合計サイズ 160 Kb の BRAM を 18Kb の BRAM9 個で構成できる。しかし、BRAM の数を変えた場合は 80 Kb の BRAM2 つに対して 18 Kb の BRAM を 5 個ずつ使用する必要があるため、消費量が増えてしまった。

BRAM のレジスタ幅を 1-bit から 3590-bit まで増やしたときの BRAM リソースの使

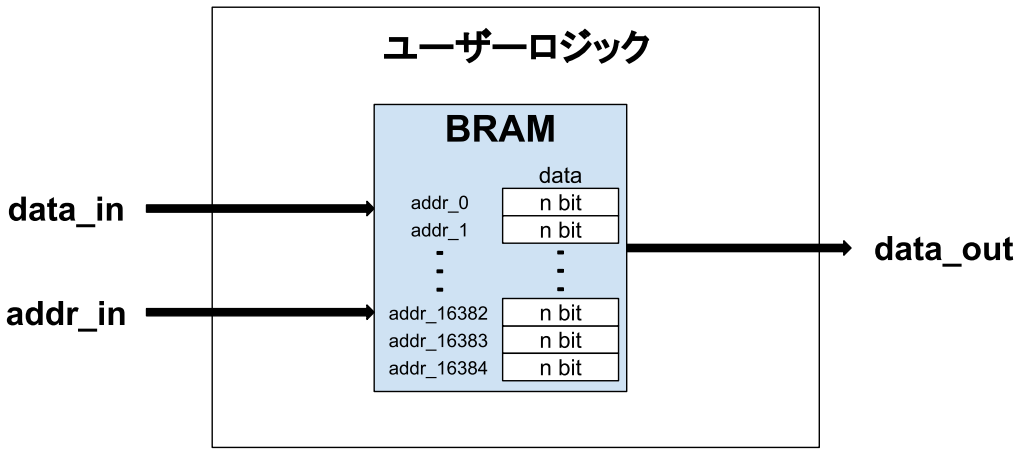


図 3.18: 搭載する BRAM のアドレス幅を変化させるときのユーザーロジックのファームウェア概略。

表 3.8: BRAM アドレスを 14-bit で、レジスタ幅を 5-bit から 10-bit に変化したときのリソース使用率。

リソース	data 5-bit	data 10-bit	
LUT	169,389	169,430	(+41)
LUTRAM	15,303	15,307	(+4)
FF	24,394	240,458	(+64)
BRAM	264	266	(+2)

用数との関係を図 3.19 に示す。BRAM の利用可能値は最大数は 1860 個であり、アドレス幅 3590-bit まで BRAM のサイズを増やせることが確かめられた。

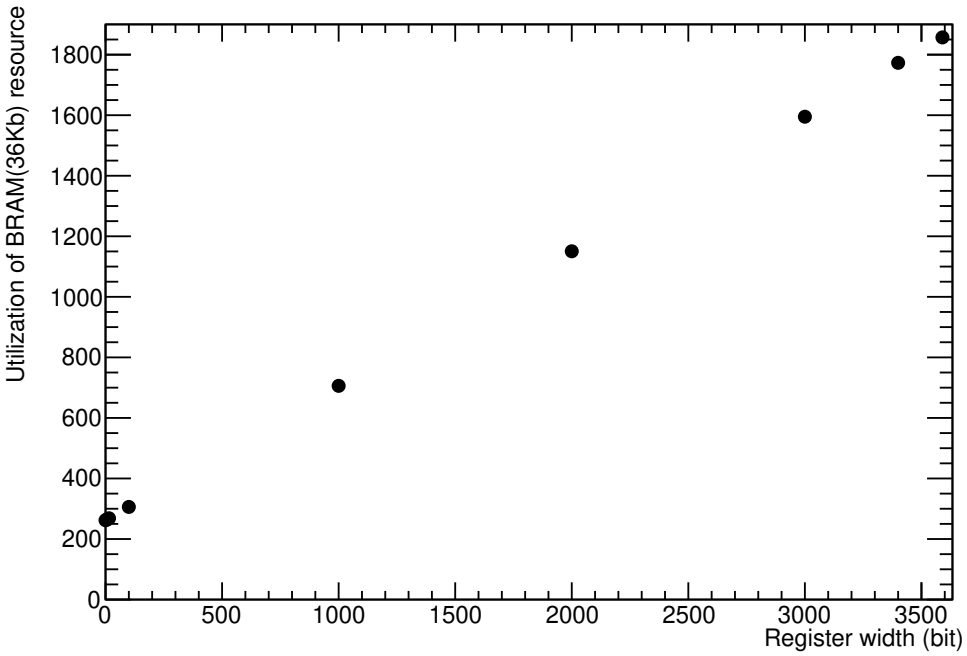


図 3.19: アドレス幅 16384 個、レジスタ幅 1-bit から 3590-bit まで変化させた時のリソース使用数の関係。

次に、BRAM のアドレス幅を変化させ、それに伴ったリソースの変化量を評価する。レジスタ幅は 5-bit で固定にしておきアドレス幅を 14-bit から 20-bit まで変化させたときの BRAM の使用率の関係を調べる。アドレス幅は最大値が固定されており、レジスタ幅に対して決められる。レジスタ幅が 5-bit のときは Xilinx IP の仕様によりアドレス幅の最大値が 20-bit に設定される。

BRAM のアドレスを 14-bit から 15-bit に変更したときのリソース使用量を表 3.9 に示す。アドレスが 14-bit のときに比べて BRAM ブロックの使用数は 2.5 個増えている。使用した BRAM のサイズはアドレスが 14-bit のときは 5-bit$\times 2^{14} = 80$ Kb となり、アドレスが 14-bit のときは 5-bit$\times 2^{15} = 160$ Kb であり、36 Kb の BRAM2.5 個分増加していることが確認できる。

BRAM のアドレスを 14-bit から 20-bit まで変化させたときの BRAM 使用数の変化を図 3.21 に示す。レジスタ幅 5-bit でアドレス 20-bit 分の BRAM サイズは 5120 Kb であり、1 ブロックあたり 36 Kb の BRAM ブロック 142.5 個を使用する。実際にこれを搭載した場合、BRAM を搭載していないときに比べて BRAM ブロックの使用数は 142.5 個増加し、計算上の見積もりと使用量が一致した。

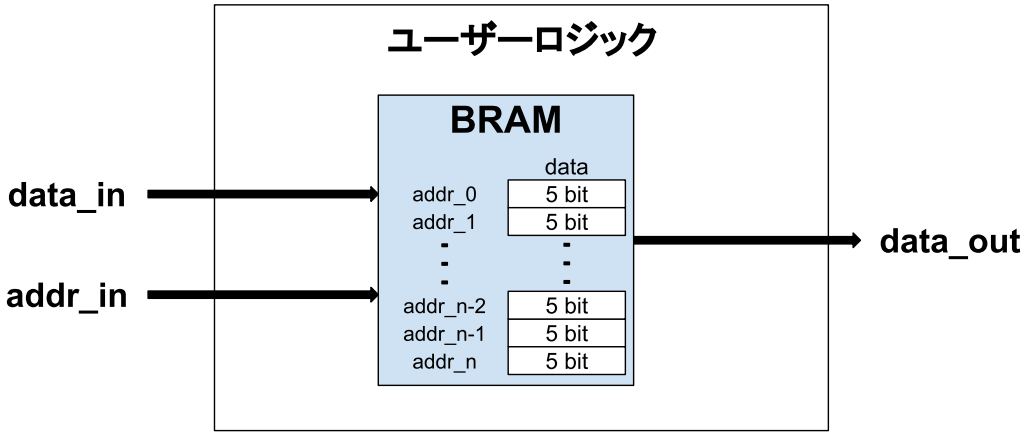


図 3.20: BRAM アドレス幅の変化に対する評価時の User IP のブロック図。

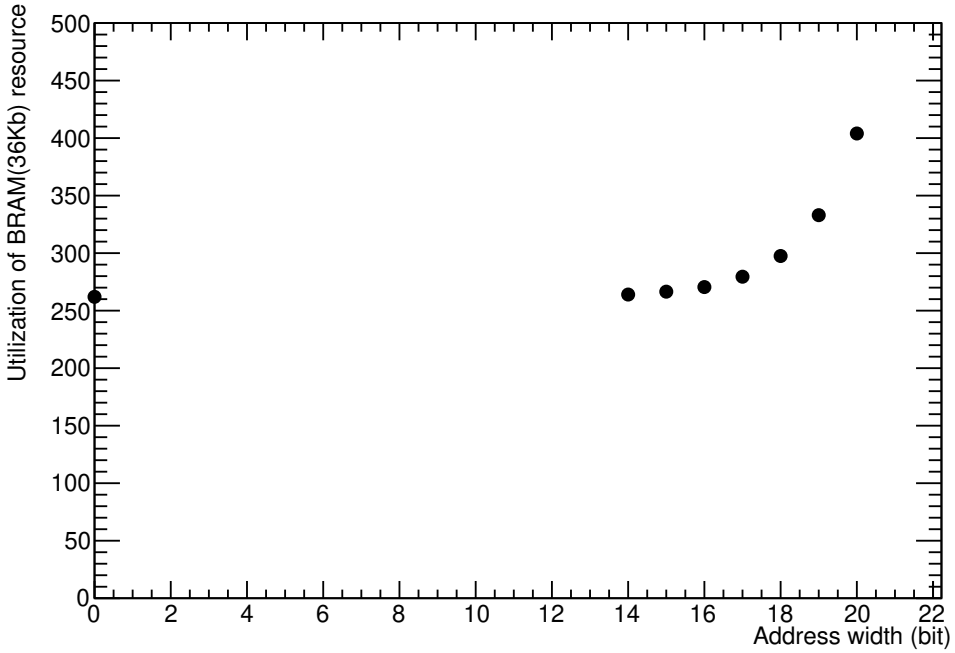


図 3.21: BRAM のレジスタ幅 5-bit、アドレスを 14-bit から 20-bit に増加させたときの BRAM ブロック使用数の推移。使用していないときに比べて、20-bit の時に 142.5 個増加しており予想通りの使用量となった。

表 3.9: BRAM のレジスタ幅 5-bit で、アドレスを 14-bit から 15-bit にしたときのリソース使用率。

	address 14-bit	address 15-bit	
LUT	169,389	169,402	(+13)
LUTRAM	15,303	15,303	(±0)
FF	240,394	240,403	(+9)
BRAM	264	266	(+2)

第 4 章

ATLAS 初段ミューオントリガーにおける開発・検証機構の利用

本章では、例として ATLAS 実験初段エンドキャップミューオントリガーのトリガーロジックを実際に実装し、リソース使用率と処理性能など、本研究で開発したシステムの評価をまとめる。

4.1 LHC-ATLAS 実験第三期運転における初段エンドキャップミューオントリガー

ATLAS 検出器の全体像を図 4.1 に示す。カロリメータを通過して、ミューオン検出器に入射する粒子は標準模型の範囲内ではミューオンとニュートリノのみで、観測可能な粒子はミューオンのみである。

衝突点で生成されたミューオンは磁場の内側の検出器を通った後にトロイド磁場領域に突入する。トロイド磁場は ϕ 方向にかかっているため、ミューオンの飛跡は η 方向に曲げられる。トロイド磁場を通り抜けたミューオンは TGC 検出器によって観測される。LHC-ATLAS 実験とそのトリガーシステムの詳細については付録 A で概説している。エンドキャップ部の初段ミューオントリガーのコンセプトを図 4.2 に示す。衝突点と TGC 検出器の情報を用いて飛跡の曲がり具合から p_{T}(横運動量) を計算し、それに閾値を設けることでトリガー判定を行う。トリガーロジックは Sector Logic (SL) ボードに実装されており、FPGA 上の BRAM を用いて参照値に p_{T} を記憶させておくことで、飛跡の曲がり具合に対する p_{T} の値を得ることができるようになっている。

TGC でのトリガー発行はトリガーセクターと呼ばれる単位ごとに行われる。トリガー

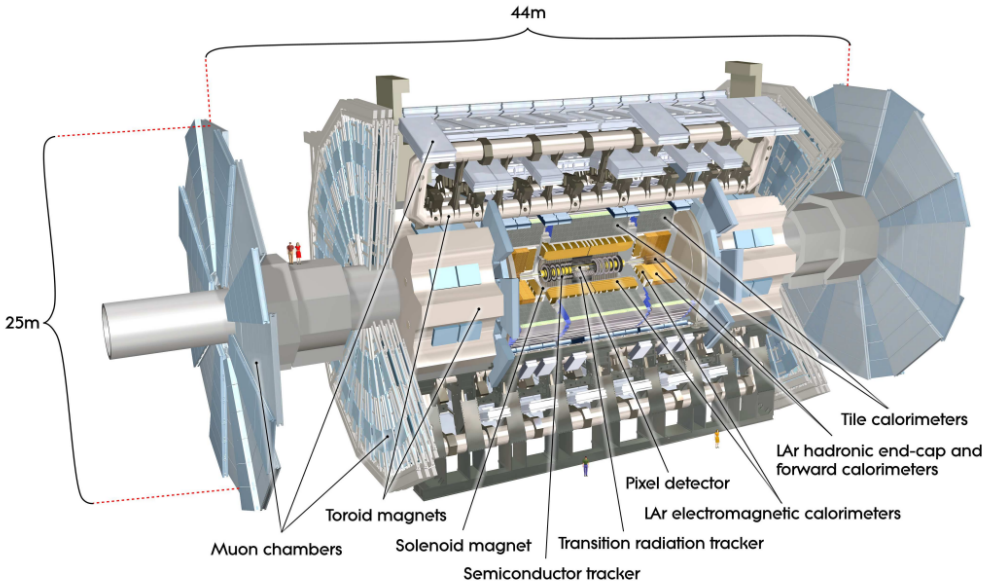


図 4.1: ATLAS 検出器の全体像 [6]。複数の検出器システムから構成されている。

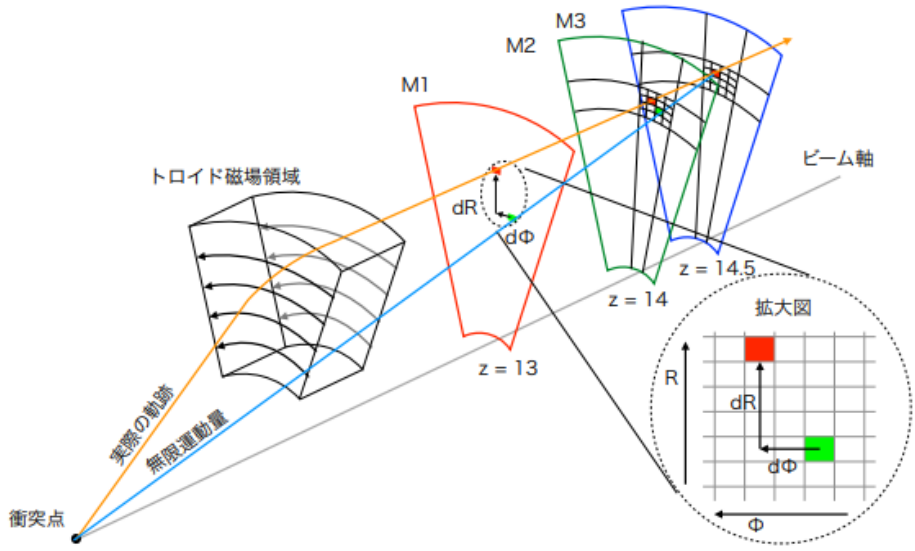


図 4.2: エンドキャップ部のミューオントリガーコンセプト [26]。M1·M2·M3 のヒット位置からミューオンの飛跡の曲がり具合を測定し、その情報を用いてトリガー判定を行う。

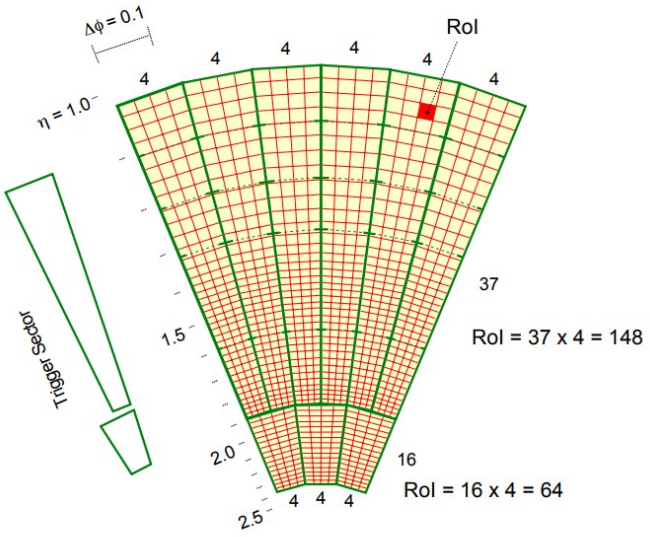


図 4.3: TGC のトリガーセクターと RoI [27]。緑の線で囲まれた領域が 1 つのトリガーセクター、赤の領域で囲まれた領域が 1 つの RoI を表す。

セクターは $1.05 < |\eta| < 1.9$ の領域を ϕ 方向に 48 分割、$1.9 < |\eta| < 2.4$ の領域を ϕ 方向に 24 分割したものである。$1.05 < |\eta| < 1.9$ の領域にあるトリガーセクターをエンドキャップ部、$1.9 < |\eta| < 2.4$ の領域にあるトリガーセクターをフォーワード部に分類する。トリガーセクターをまたぐ情報の共有は行われない。

図 4.3 に示すように 1 つのトリガーセクターは η, ϕ 方向に分割され、Region of Interest (RoI) という単位に分割される。エンドキャップトリガーセクターの RoI は 1 トリガーセクターを η 方向に 37 分割、ϕ 方向に 4 分割したもの (合計 148 RoI) であり、大まかに $\Delta\eta \times \Delta\phi = 0.02 \times 0.03$ に対応する。RoI が初段ミューオントリガーの最小単位であり、これより細かい分解能の情報は初段トリガーでは用いることができない。本研究では数トリガーセクター分のトリガー処理を User IP に実装し、何台分の SL の処理を同時に検証できるのか等、リソースの観点から調査し検証する。

4.2 初段ミューオントリガーの実装

User IP 部分はトリガーロジックを入れる部分であり、図 3.6 のようにファームウェアを LHC-ATLAS 実験 Run-3 の SL のファームウェアに入れ替えた。開発したコアの動作クロックは 300 MHz であるのに対し、SL の動作クロックは 40 MHz であるため、

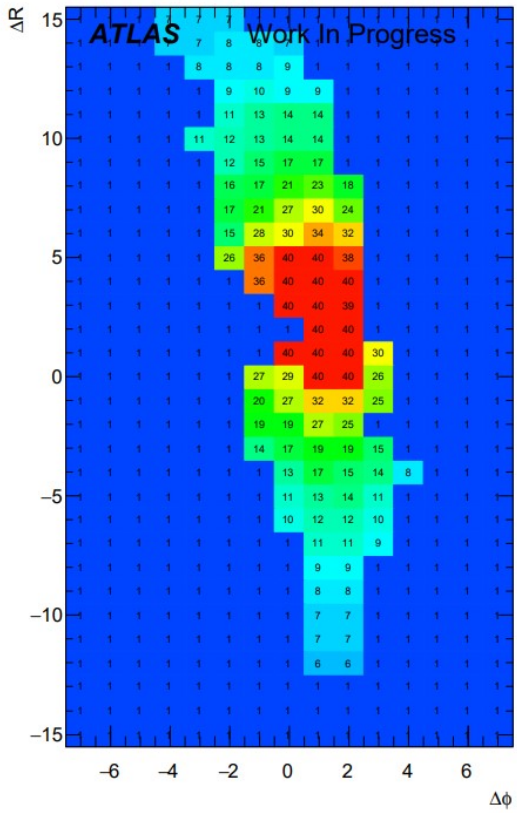


図 4.4: トリガーに使用される参照値 [28]。

FPGA コアにおける動作クロックの変更を加えた。変更は vitis ツールを用いたプラットフォームリンク時に`--clock` オプションを使用して変更した。

ホストアプリケーションは 3.2 節と同じである。使用したコインシデンス参照値は ATLAS 実験 Run-3 のファイルで 1 トリガーセクター分の値が格納されている。BRAM の入力に使用したコインシデンス値を図 4.4 に示す。ΔR と $\Delta\phi$ から p_{T} の値を決めるコインシデンスで、この参照値をもとにトリガー判定を行える。

4.3 ATLAS-Run3 初段エンドキャップミューオントリガーを用いたファームウェア開発・検証機構の評価

User IP のファームウェアに実装したトリガーロジックの処理を図 4.5 に示す。トリガーロジックはワイヤー、ストリップからの入力情報をもとに p_{T} 判定を出力する。本節では、ファームウェア開発・検証機構を評価するために、ΔR、$\Delta\phi$ が小さい高い p_{T} が

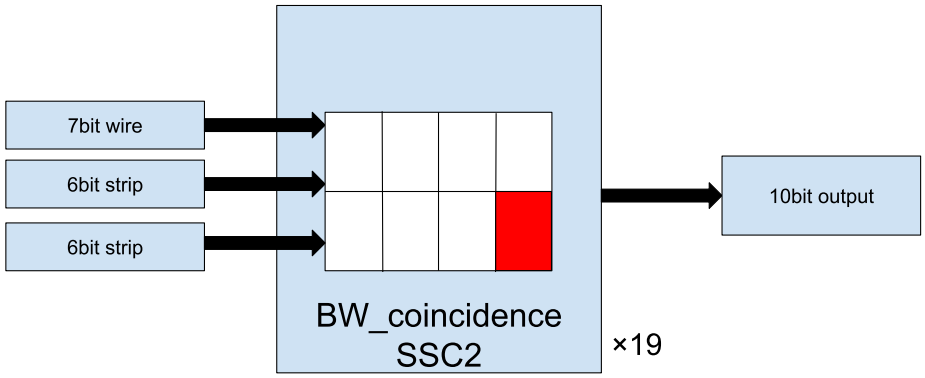


図 4.5: SL における SSC2 の処理。検証機構の評価をするために、図の赤枠 RoI を通る ΔR,$\Delta\phi$ が 0 の HPT コインシデンスがとれた正の電荷の信号を入力した。

表 4.1: トリガー判定に用いる入力信号のフォーマット。

bit	6	5	4	3	2	1	0
wire	POS	H/L	sign	ΔR			
strip	nothing	POS	H/L	sign	$\Delta\phi$		

予想されるミューオンの情報を入力値にして、適切なトリガー出力が得られるか評価した。トリガー出力が得られるのは 148RoI のうち、1 つの RoI のみが期待される。入力は表 4.1 に示したような ΔR 情報を含んだワイヤー情報 7-bit、$\Delta\phi$ 情報を含んだストリップ情報 6-bit を入力した。pos 情報を用いて SSC 内の RoI 位置を決定でき、H/L 信号により前段の HPT ボードでコインシデンスが取れたかがわかる。1RoI のみコインシデンスを取れたデータを入力し、他の RoI には値 0 を入力した。

4.3.1 トリガー検証評価

トリガー処理の出力結果を表 4.2 に示す。p_{T} 閾値は 4-bit で 15 段階で設定されている。出力結果は 15 であり、p_{T} が 20 GeV 以上である判定結果が得られた。RoI は得られたミューオンの飛跡が通過した RoI の位置を示す。入力した検出器位置の RoI から判定結果が得られた。7 bit 目は電荷の情報で、ミューオンの飛跡の曲がった方向がわかる。8 bit 目は HPT ボードにおいて Wire, Strip ともにコインシデンスがとれていたかを出

表 4.2: 入力信号のトリガー処理によって得られたトリガー判定値。

bit	9	8	7	6	5	4	3	2	1	0
	その他		電荷	RoI			p_{T} 閾値			

表 4.3: p_{T} が高いミューオンイベントを検出器の 1 つにあるような入力を与えたときの、トリガー処理結果。

項目	time (ms)
Alveo ファームウェア読み込み	16
Alveo デバイスの読み込み	279
データ準備	110
バッファの準備	20
PC から Alveo への DMA 転送	13
FPGA コアの処理	115
Alveo から PC への DMA 転送	10
その他	7
合計	570

力する。9 bit 目はバックグラウンドを削減するためのフェイクミューオンであるかどうかを示すビットである。この結果からは 20 GeV を超える p_{T} のミューオンイベント候補が出力できていることが分かった。トリガー検証が正常に行えることが分かったので、処理レートの評価とリソースの評価を行う。

4.3.2 処理レートの評価

今回用いた入力イベント数は 4,194,304 個 (転送レートの上限) でその処理時間の結果を表 4.3 に示す。全体の処理時間は 570 ms である。もっとも重要な処理時間は PC から Alveo への DMA 転送・FPGA コアの処理・Alveo から PC への DMA 転送の 3 つの合計時間である。この 3 つの処理を繰り返すことで、検証をループさせることができる。合計時間は 138 ms であり 30 MHz で検証を行えることが分かった。ソフトウェア処理レートに比べて十分な検証速度による動作が確認できた。

表 4.4: トリガーロジックを実装した際のリソース使用量。

項目	コアのみ	1 トリガーセクター
BRAM	262	430
LUT	169,419	173,259
FF	240,493	244,883
MMCM	3	6

4.3.3 リソースの評価

トリガーロジックを実装した際のリソース使用率を表 4.4 に示す。何も実装していないときと 1 トリガーセクターを実装した時で、LUT, FF, LUTRAM の使用量はワイヤーやレジスタの配線の変更によって消費量が少し増えている。また、MMCM はクロックの変更に利用されるので、40MHz クロックの作成とトリガー内部で生成する 2 種類のクロックに使用され合計 3 つ使われる。最も大きく変化したリソース使用率は BRAM の値で、168 個消費されトリガーセクター数を増やすことで最初に限界が来ると予想した。計算ではトリガーセクター 15 枚分のトリガーロジック搭載が可能であるので、その限界を調べた。トリガーセクターの実装数とリソース使用率の関係を図 4.6 に表す。トリガーセクター 2 枚以上搭載したときの BRAM リソースの使用数も 100 個ずつ増えており、15 枚分のトリガーセクター処理実装が限界であることが分かった。1 枚のセクターロジックボードは 2 トリガーセクター分の処理を行うため、使用可能な BRAM のリソースは 1600 個であるから、SL ボード 7.5 枚分 (15 トリガーセクター) まかなうことができる。SL によるエンドキャップのトリガーセクター処理は全部で 96 枚であるので ATLAS エンドキャップ SL の 7.5/96 検証可能である。

特定の入力信号ではあるが、ファームウェア開発・検証機構を用いたトリガーロジックの検証に成功した。ATLAS 初段ミューオントリガーを実機上以外で実装した初めての検証であり、本検証機構を用いることでトリガー検証を行えることができ、高レートでの処理が行えることが分かった。

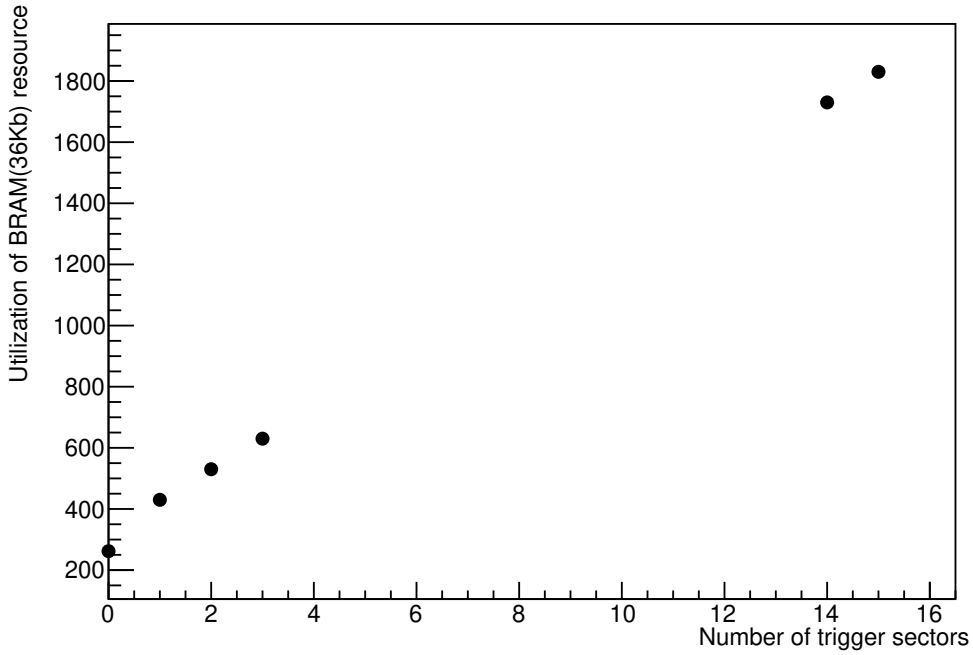


図 4.6: トリガーセクターの数に対するリソース使用率。1 トリガーセクターの実装には BRAM170 個、2 トリガーセクター以上の実装には BRAM100 個ずつ使用される。

第 5 章

結論と今後の展望

素粒子実験では標準模型の精密測定や新物理の発見を目指し、さらなる実験の精度向上を目指している。実験精度の向上のためには統計量を増やすことが求められ、加速器の高輝度化 · 高エネルギー化や検出器の大型化 · 精密化が行われている。これに伴い、実験で取得するイベントレートも増加し、レートの削減のためのトリガー処理が重要な役割を担う。トリガーは FPGA に論理回路として書き込むことで処理を行っており、加速器の高輝度化 · 高エネルギー化や検出器の大型化 · 精密化に伴って論理回路が複雑化 · 大型化している。本研究ではトリガー開発の効率化を行うために、Alveo データセンターアクセラレータカードを用いてトリガーロジック開発 · 検証機構をデザインした。

Alveo は FPGA 搭載型のアクセラレータカードであり、PC で行う処理の一部を Alveo 上の FPGA で行うことができる。そこで、ユーザーが自由に書き換えることができる Dynamic Region にデータ転送のためのコアを作成し、トリガーロジックに置き換える User IP 部分を作成した。コアの動作確認として User IP 部分に BRAM を実装した。BRAM に参照値を記憶させ、その値が正しく書き込まれることを確認することで、ファームウェア開発・検証機構のデザインが正しく動作していることを確認した。

また、例として ATLAS 実験 Run-3 の初段エンドキャップミューオントリガーを用いて構築したファームウェア開発 · 検証機構を利用した。User IP の部分のロジックを入れ替えることで、トリガーの動作が正常に行えるかを確認し、リソース使用量と処理性能の評価を行った。搭載できたトリガーロジックは ATLAS 実験の SL7.5 枚分であり、30 MHz の処理レートで検証できることが分かった。また、特定のイベントを用いたトリガーロジックの検証に成功し、ATLAS 実験のトリガーファームウェアの実装に成功した。開発したファームウェア開発・検証機構はトリガー検証を行える十分な評価が得られた。

今後の目標としては、シミュレーションソフトを走らせることでシミュレーションイベ

ントを生成してトリガー検証を行い、様々なイベントに対してトリガー検証を行っていきたい。また、近年のトレンドである機械学習などをを用いたトリガー開発を行える環境であるので、検証と同時にトリガーの開発にも利用する。すなわち、本検証機構を利用して様々な FPGA ファームウェアの開発を行っていくことが期待される。

付録 A

ATLAS 実験-Run3 における初段エンドキャップミューオントリガー

本章では第 3 章でデザインしたファームウェア開発 · 検証機構に第 4 章で例として ATLAS 実験-Run3 で使われている Sector Logic のトリガーロジックを実装してリソース使用率と処理性能を評価した。

A.1 ATLAS 実験 Run-3 初段ミューオントリガー

A.1.1 トリガー用ミューオン検出器

ミューオン検出器は ATLAS 検出器の最外層に位置する。LHC-ATLAS 実験における Endcap 部のミューオントリガーは TGC 検出器が担っている。TGC は図 A.1 に示したような MWPC の構造をしており、アノードワイヤーとカソードストリップが直行するように配置されているため、2 次元読み出しが可能になっている。アノードワイヤーには 2.9 kV の高電圧が印加されており、ガスギャップ中の CO_2 : n-pentane = 55 : 45 混合ガスの電離によって生じた電子をワイヤー電極とストリップで読み出し、荷電粒子の通過位置を測定する。ガスギャップ及びワイヤー間隔が小さいことで検出器の時間応答は早く、特にワイヤーでは 90% 以上の確率で粒子が通過してから 25 ns 以内に信号が得られる。

図 A.2 に示すように、ATLAS 検出器では 2 層 1 組 (doublet)、3 層 1 組 (triplet) の TGC を用いて、各層とのコインシデンスを取ることにより、フェイク信号を削減している。TGC 検出器の配置を図 A.3 に示す。TGC は磁場領域の内側に 1 つ、外側に 3 つの合計 4 つで構成されている。磁場の内側のステーションは η 方向に分割された 2 つのチェ

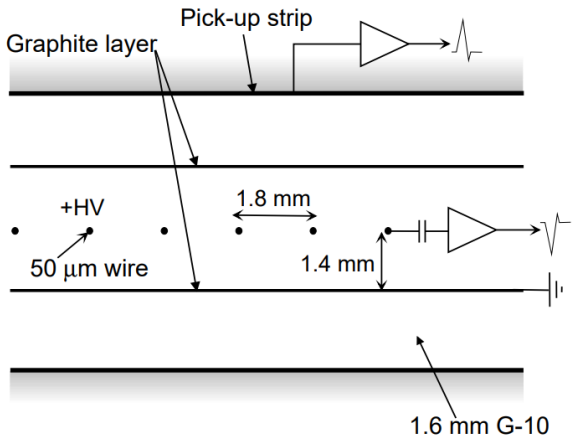


図 A.1: TGC 検出器の構造 [6]。ガスギャップ 2.8 mm、直径 50 µ m のアノードワイヤーが 1.8 mm の感覚で貼られている MWPC(Multi Wired Proportional Chamber) の構造をしている。

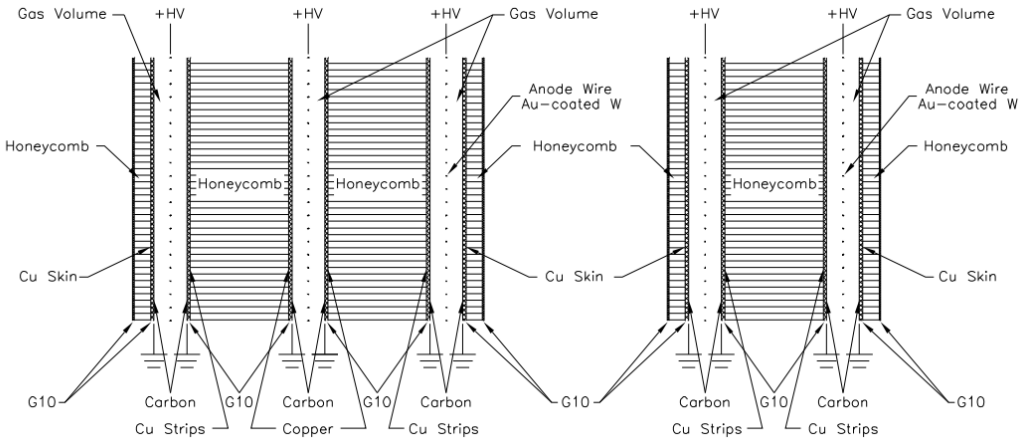


図 A.2: TGC 検出器の doublet·triplet の断面図 [6]。各ガスギャップの間はハニカム構造のパネルで隔てられている。

ンバーから構成され、$|\eta|$が小さい (R の大きい) 法のチェンバーを EI チェンバー、$|\eta|$が大きいほうのチェンバーを FI チェンバーと呼ぶ。磁場の外側のステーションの 3 層は衝突点に近い側から順に M1, M2, M3 と呼ぶ。EI/FI、および M2、M3 は doublet チェンバーであり、M1 は triplet チェンバーである。EI/FI と M1,M2,M3 の間に磁場がかかっており、図 A.3 に示したように低い横運動量をもつミューオンは大きく曲げられ、高い横運動量をもつミューオンはあまり曲げられない。

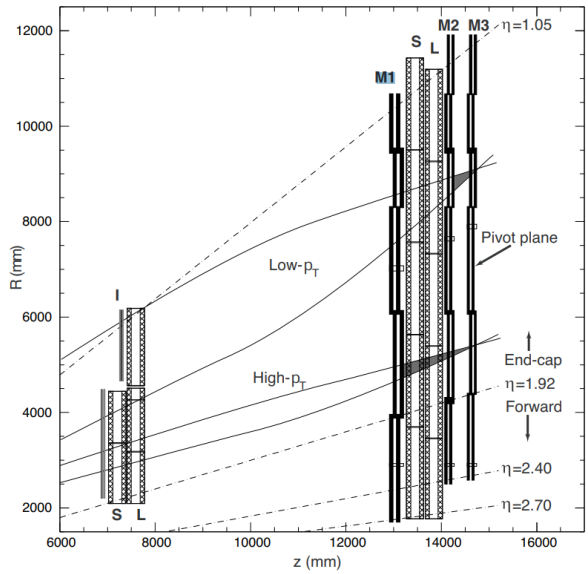


図 A.3: TGC の配置 [6]。磁場の内側に EI 及び FI チェンバー、外側に M1、M2、M3 が設置されている。

A.1.2 トリガー単位

TGC でのトリガー発行はトリガーセクターと呼ばれる単位ごとに行われる。トリガーセクターは $1.05 < |\eta| < 1.9$ の領域を ϕ 方向に 48 分割、$1.9 < |\eta|$ の領域を ϕ 方向に 24 分割したものである。以下では、$1.05 < |\eta| < 1.9$ のものを Endcap トリガーセクター、$1.9 < |\eta|$ では Forward トリガーセクターと呼ぶことにする。トリガーセクターをまたぐ情報の共有は行われない。

図 A.4 1 つのトリガーセクターは η、ϕ 方向に分割され、Region of Interest（RoI）という単位に分割される。Endcap トリガーセクターの RoI は 1 トリガーセクターを η 方向に 37 分割、ϕ 方向に 4 分割したものであり、大まかに $\Delta\eta \times \Delta\phi = 0.02 \times 0.03$ に対応する。RoI が初段ミューオントリガーの最小単位であり、これより細かい分解能の情報は L1 では用いることができない。

A.1.3 初段 Endcap ミューオントリガー

ATLAS 実験における陽子バンチ衝突頻度は 40 MHz である一方、データとして記録できるイベントレートは約 1 KHz である。この制約により、ATLAS 実験では図 A.5 に示

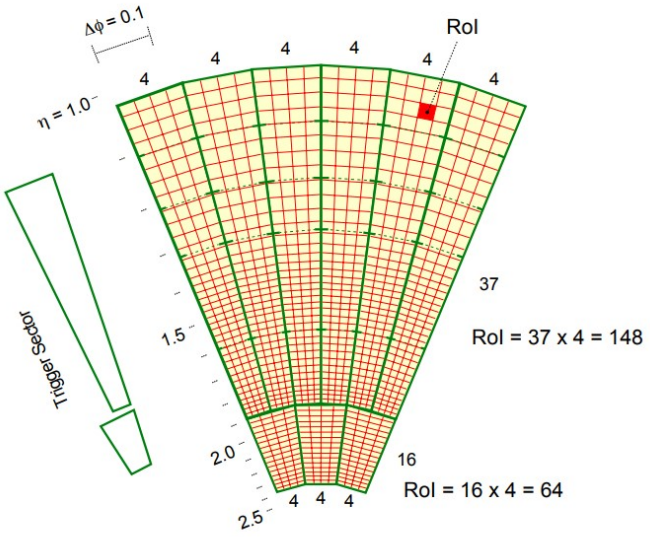


図 A.4: TGC のトリガーセクターと RoI [27]。緑の線で囲まれた領域が 1 つのトリガーセクター、赤の領域で囲まれた領域が 1 つの RoI を表す。

すように初段トリガーと後段トリガーの 2 段階でトリガーをかけている。イベントレートは初段トリガーで 2.5 μs 以内に 40 MHz から 100 kHz まで絞られ、後段トリガーによって数秒以内に 1 kHz までレートを削減し、物理データを保持する。

初段トリガーでは、図 A.5 に示すように 2.5 μs 以内に 40 MHz で送られてくる全イベントに対してトリガー判定を行い、100 kHz にイベントレートを選別する必要がある。これを満たすために、初段トリガーシステムは高速処理が可能な ASIC 及び FPGA などの集積回路からなるハードウェアで実装されている。FPGA とは、Field Programmable Gate Array の略称で、使用者が中の論理回路を自由に書き換えることができる集積回路である。L1 システムがトリガー判定を行っている間、データは検出器上のフロントエンド回路（FE）の Buffer に保持されている（L1 Buffer）。

L1 はカロリメータの情報を用いて発行される L1Calo、ミューオン検出器の情報を用いて発行される L1Muon、それらを組み合わせた複合的なトリガー L1Topo の 3 種類に分類される。ミューオンのトリガーは検出器のバレル部分とエンドキャップ部分で別々に判定され、それらの情報を MUCTPI（Muon to CTP Interface）に送り、まとめた後にトリガー判定を行う。L1Calo と L1Muon はそれぞれ Central Trigger Processor（CTP）に送られると同時に、Topology Processor（L1Topo）に送られる。L1Topo では L1Muon と L1Calo の情報を組み合わせて、それぞれの数や位置関係から複合的な L1 トリガーを

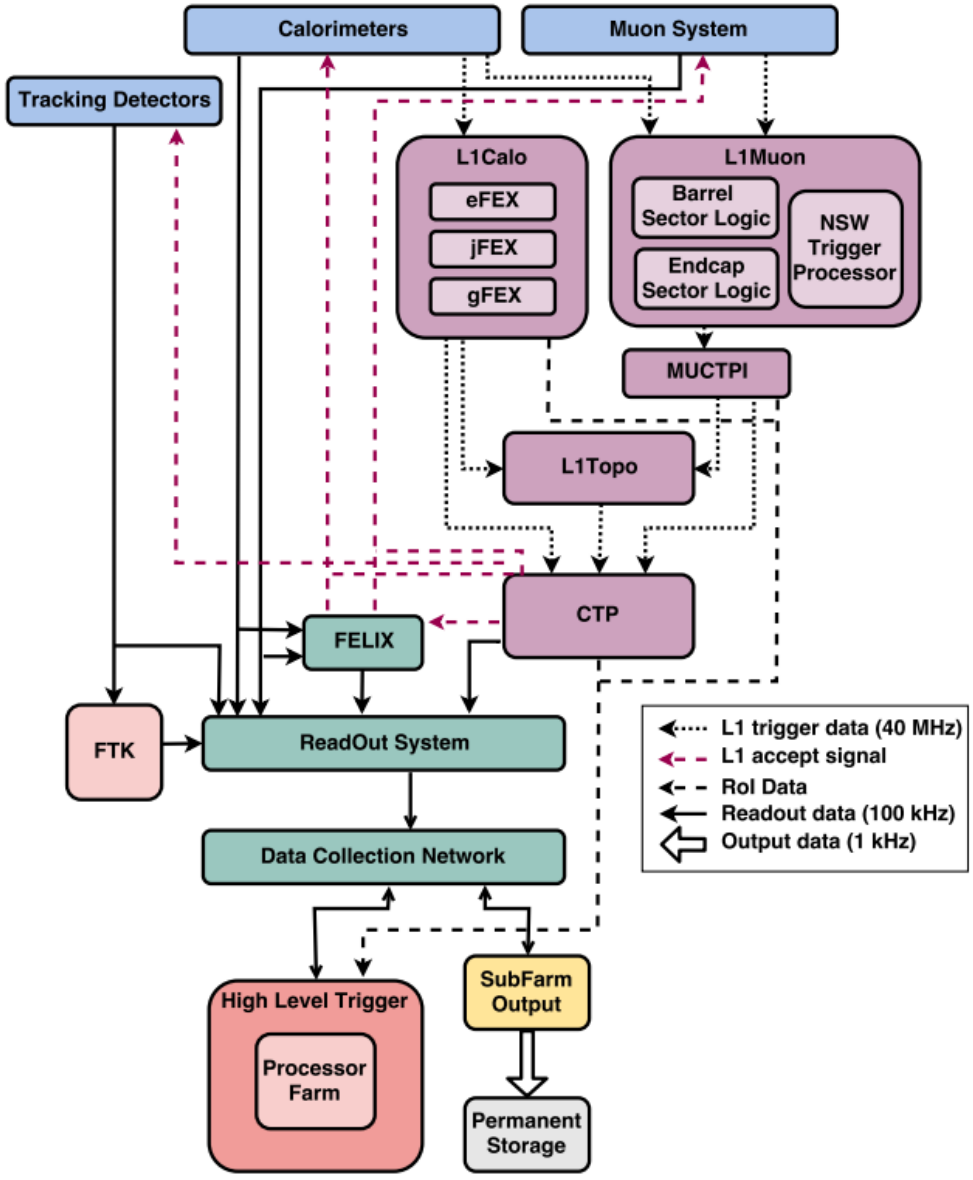


図 A.5: LHC-ATLAS 実験 Run3 におけるトリガー・データ取得システムの概要 [29]。

発行し、CTP に送られる。CTP に集められたあと、100 kHz に収まるようにプリスケールをかけられた後に L1Accept（L1A）としてトリガー発行される。

表 A.1: HPT ボードから SL に送信するデータのフォーマット。$\Delta R(\Delta\phi)$ の情報は符号も含めて 4bit(3bit) で送信される。H/L は HPT コインシデンスがとれたかどうかの情報で、1bit で送信される。POS 及び HIT ID はヒット位置に関する情報で 4bit で送信される。

Bit	9	8	7	6	5	4	3	2	1	0
wire	HITID			POS	H/L	Sign	ΔR			
strip	Not used	HITID			POS	H/L	Sign	$\Delta\phi$		

A.1.4 トリガーロジック

TGC 検出器を用いたトリガー判定は Sector Logic (SL) と呼ばれるモジュールで行われる。PP ASIC から送られた TGC-BW のワイヤー、ストリップのヒット情報を用いて、SLB ASIC の中でトリプレット、ダブレットごとにコインシデンスを取る。SLB ASIC のコインシデンス結果をもとに HPT (High-PT) board と呼ばれるモジュールでトリプレット、ダブレットをまたいだコインシデンスを取り、SL にデータを送信する。HPT ボードと SL との間の通信には G-Link と呼ばれる通信技術が用いられる。G-Link は信号を光信号としてシリアライズし、光ケーブルで送信する。1 本の光ケーブルは 16bit または 17bit を送信する。クロックは 40 MHz で信号を送り、各ケーブルでの信号の伝送速度は $16(17)\times$ 40 M $= 640(680)$ Mbps である。

1 つのトラック情報はワイヤーからは 10bit、ストリップからは 9bit で入力され、その内訳は表 A.1 のようになる。

SL ファームウェアのトリガーロジックを図 A.6 に示す。SL ボード 1 枚で 2 つのトリガーセクターのトリガー処理を行い、それぞれ SSC 単位で処理が行われる。セクターロジックのトリガー判定は TGC-BW の δR,$\delta\phi$ から、LUT を用いて p_{T} を算出している。第三期運転からは新しく追加された検出器である NSW などとインナーコインシデンスを取り、背景事象をより削減しトリガー判定を出している。

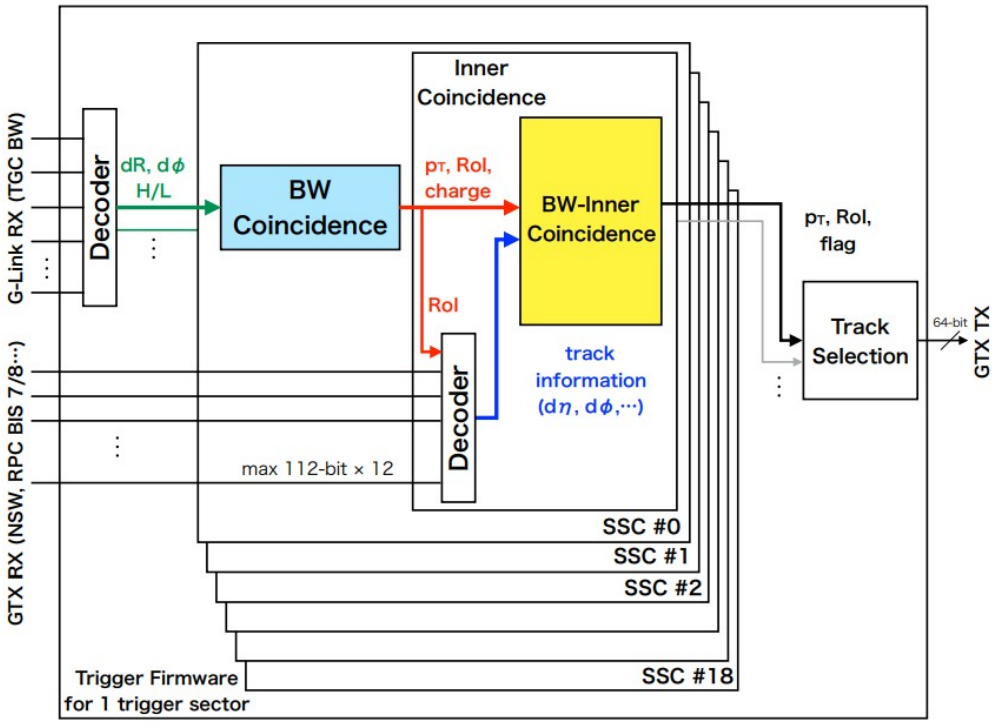


図 A.6: SL のファームウェア [27]。

付録 B

用語集

AXI4: Advanced eXtensible Interface 4 の略。ARM 社が提供する 4 世代目の AMBA インターフェイス規格。

BRAM: Block Random Access Memory の略。FPGA 内に大量のデータを格納するために使用されるメモリ。アドレス幅やレジスタ幅を自由に設定できる。SRAM との違いは書き込みの制御をお超えたり、デュアルポートでの読み書きが行える機能があるところである。FPGA チップ上で使用できる BRAM の数は限られており、メモリサイズは 4/8/16/32Kb が一般的。本研究で使用している Alveo U200 の FPGA では 36Kb サイズの BRAM が搭載されている。

DDR SDRAM: Double Data Rate Synchronous Dynamic Random Access Memory の略。パソコンに搭載されている一時記憶装置。ハードディスクや SSD からデータを一時的に呼び出して記憶し、作業をするための装置。本研究で使用している PC と Alveo に搭載されている。

DMA: Direct Memory Access の略。コンピュータシステム内のデータ転送方式の 1 つ。CPU を介さずに周辺機器やメインメモリなどの間で直接データ転送を行う方式。

Dynamic Region: Alveo データセンターアクセラレータカードの FPGA 内の機能を分けたときの 2 つの領域の内の 1 つ。LUT, FF, DSP, BRAM などを用いた配置配線を行うことで CPU で行う処理の一部を高速に処理するための機能を自由に搭載できる。この領域でトリガーロジックを自由に入れ替えることができるような設計を行える。

hls4ml: 一般的に出回っているライブラリで学習した機械学習を C/C++ 言語に変換するライブラリ。Vivado HLS を用いて、FPGA に搭載できる形に高位合成できる。

MMCM: Mixed-Mode Clock Manager の略。ある入力クロックに対して定義済みの位

相及び周波数を持つ複数のクロックを生成する。

SRAM: Static Random Access Memory の略。FPGA 内にデータを格納するためのメモリ。アドレス幅やレジスタ幅を自由に設定できる。

Static Region: Alveo データセンターアクセラレータカードの FPGA 内の機能を分けたときの 2 つの領域の内の 1 つ。DMA や AXI インターフェースが搭載されており、これを用いることでホストとデバイスの間でデータ転送ができるようになる。その他にも FPGA コンフィギュレーションやデータ転送の監視機能などが搭載されている。

URAM: Ultra Random Access Memory の略。各ブロック 288Kb のブロックメモリを提供し、ブロック同士を接続することで大規模なオンチップストレージを構築できる。

Vivado: AMD Xilinx 社の FPGA の統合型設計環境。回路図とテキスト形式によるデザイン入力、統合された VHDL と Verilog HDL 合成、配置配線、タイミング検証、プログラミングなどの機能を兼ね備えている。

Vivado HLS: AMD Xilinx 社による高位合成ツール。C 言語を用いたソースコードから AMD Xilinx 社の FPGA 用の IP 及び RTL 記述を生成できる。

Vitis: AMD Xilinx 社によるソフトウェア言語を用いて FPGA などのハードウェアプラットフォーム上の開発を可能にする開発環境。ハードウェアの知識がなくても FPGA を設計できる。

謝辞

本研究を進めるにあたりたくさんの方に支えていただきました。そのすべての方々に深く御礼申し上げたいと思います。指導教員の前田順平先生からは、様々なことを学ばせていただきました。ATLAS においては、トリガーから検出器のことまで幅広い知識をご教授いただきました。また、ハードウェアの知識についてもご教授いただいて研究の進展をサポートしていただいたことに大変感謝しております。本論文の作成に関しても、私の拙い文章を細かく細かく添削していただきありがとうございました。神戸 ATLAS グループの藏重久弥先生、山﨑祐司先生、越智敦彦先生には日々の研究生活やミーティングにおいてたくさんのご指導をしていただきました。本当にありがとうございました。名古屋大学の堀井泰之先生には機械学習や FPGA に関した先行研究についてご教授いただきました。

同研究室の竹内康雄先生、身内賢太朗先生、鈴木州先生、東野聡先生には論文講究や講義などで丁寧なご指導をいただきました。本当にありがとうございました。もう卒業されましたが、研究室の大先輩である日比宏明さん、水越彗太さん、石浦宏尚さんからは様々な知識を学ばせていただきました。本当にありがとうございました。日比さん、ATLAS に関するわからないことを丁寧にわかりやすく教えてくださり、ありがとうございました。水越さん、プログラミングの質問に迅速にこたえてくださりありがとうございました。石浦さん、見てるだけで心が落ち着きました。ありがとうございました。

ATLAS グループの一つ上の先輩である寺村七都さん、安部草太さん、野口健太さん、池森隆太郎さん、末田晧介さんには大変お世話になりました。寺村さん、池森さん、野口さん、一緒に散歩してくださりありがとうございました。寺村さん、歩くのが遅くて心配してました、足は健康ですか。池森さん、散歩に誘ってくれる時の笑顔が素敵でした。野口さん、散歩をやめてしまった時が少し悲しかったです。安部さん、散歩こそできなかったですが丁寧に質問に答えてくださりとても支えになりました。ありがとうございました。ATLAS グループではない先輩の谷口大吾さん、Kostar Yurii さん、長崎大智さん、前田剛志さん、窪田諒さん、大変お世話になりました。ありがとうございました。窪田さ

ん、飲みに誘ってくれたり、麻雀をご教授いただきました。ありがとうございました。

同期の中村竜也君、金崎奎君、中山郁香さん、高橋真斗君、山下翼君、皆がいたおかげでここまで研究をやり遂げることができました。中村君、ともに ATLAS グループとしてわからないことを教えてくれたり、本当にありがとう。ATLAS 以外の同期の人たちも本当にありがとう。また一つ下の後輩である田路航也君、森本晴己君、山下智愛さん、濱田悠斗君、鐘海文君、大藤瑞乃さん、高木優祐君、安博充君、先輩らしいことは何もできませんでしたが、これからの粒子物理研究室を引っ張ってくれることを祈っております。森本君、田路君、散歩してくれて本当にありがとう。森本君にはコアタイムを最後までやり遂げてもらいたいです。大学院生活の約 2 年間、コロナの影響で大変なこともありましたが、たくさんのことを学び充実した日々を送ることができました。関わっていただいた皆様のおかげです。ありがとうございました。そして最後に、24 年間ここまで育ててくれた両親に深く深く感謝し、これから社会人として恩返ししていくことを誓います。

参考文献

[1] Yuki Akimoto, higgstan.com https://higgstan.com/standerd-model/.

[2] A. Pomarol, Beyond the Standard Model, arXiv:1202.1391.

[3] Belle II experiment https://www.belle2.org/.

[4] スーパーカミオカンデ 公式ホームページ
https://www-sk.icrr.u-tokyo.ac.jp/sk/.

[5] The Large Hadron Collider | CERN
https://www.home.cern/science/accelerators/large-hadron-collider.

[6] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, Journal of Instrumentation **3** (2008) S08003.

[7] G. Aad *et al.*, Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC, Phys. Lett. B **716** (2012) 1–29.

[8] Y. Fukuda *et al.*, Evidence for Oscillation of Atmospheric Neutrinos, Phys. Rev. Lett. **81** (1998) 1562–1567.

[9] 安部草太, 高輝度 LHC-ATLAS 実験に向けた深層学習を応用したトリガーシステムの提案と性能評価, 神戸大学 修士論文 (2022).

[10] P. Calafiura, J. Catmore, D. Costanzo, A. Di Girolamo, ATLAS HL-LHC Computing Conceptual Design Report, 2020.

[11] ATLAS Software and Computing HL-LHC Roadmap https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/UPGRADE/CERN-LHCC-2022-005/.

[12] C. Sun *et al.*, Fast muon tracking with machine learning implemented in FPGA, Nucl. Instrum. and Meth. A **1045** (2023) 167546.

[13] AMD Xilinx, UltraScale Architecture and Product Data Sheet (DS890).

[14] AMD Xilinx, Virtex-II Platform FPGAs:Complete Data Sheet (DS031).

[15] AMD Xilinx, 7 Series Product Tables and Product Selection Guide https://japan.xilinx.com/content/dam/xilinx/support/documents/selection-guides/7-series-product-selection-guide.pdf#K7.

[16] ATLAS Athena Guide | Athena Introduction https://atlassoftwaredocs.web.cern.ch/athena/athena-intro/.

[17] AMD Xilinx, Getting Started with Alveo Data Center Accelerator Cards User Guide (UG1301).

[18] AMD Xilinx, Alveo U200 Data Center Accelerator Card https://www.xilinx.com/products/boards-and-kits/alveo/u200.html.

[19] 株式会社 HPC テック | XILINX ALVEO Series https://www.hpctech.co.jp/hardware/xilinx-alveo-series.html.

[20] Wikipedia - PCI Express https://ja.wikipedia.org/wiki/PCI_Express.

[21] AMD Xilinx, DMA Configurations · Alveo Data Center Accelerator Card Platforms User Guide (UG1120).

[22] AMD Xilinx, Vivado-AXI-Reference-Guide (UG1037).

[23] AMD Xilinx, Vitis Application Acceleration (UG1393).

[24] 末田晧介, Zynq を用いた FPGA ロジック監視システムの構築, 神戸大学 修士論文 (2022).

[25] AMD Xilinx, Xilinx Native API https://xilinx.github.io/XRT/master/html/xrt_native_apis.html.

[26] 赤塚駿一, LHC-ATLAS 実験 Run-3 に向けたミューオントリガーの改良, 京都大学 修士論文 (2017).

[27] Junpei Maeda *ed.*, Full design report of the ATLAS Level-1 Endcap Muon Trigger in the Phase-I upgrade, ATL-DA-ER-0001.

[28] 塩見崇宏, LHC-ATLAS 実験における第三期運転に向けた初段ミューオントリガーアルゴリズムの開発, 神戸大学 修士論文 (2021).

[29] ATLAS Collaboration, Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System, 2017, CERN-LHCC-2017-020, ATLAS-TDR-029.