修士学位論文

LHC-ATLAS実験 液体アルゴンカロリメータのデジタルトリガー：重イオン衝突の低エネルギー事象のためのトリガー発行アルゴリズムとファームウェアの開発

(LHC-ATLAS Experiment Digital Trigger for Liquid Argon Calorimeter: Trigger Algorithm and Firmware Development for Low-Energy Events in Heavy Ion Collisions)

東京大学大学院
理学系研究科 物理学専攻
田中研究室
西村ダニエル

2025年1月26日

目 次

図 目 次

表 目 次

概 要

欧州原子核研究機構 (CERN) が保有する Large Hadron Collider (LHC) では、標準模型の検証や超対称性粒子の探索などを目的とした陽子・陽子衝突実験に加えて、宇宙誕生直後に存在したとされる Quark-Gluon Plasma (QGP) の研究などを目的とした重イオン衝突実験が実施されている。この重イオン衝突実験では、ATLAS 検出器でもデータ取得が行われており、QGP の生成過程やその特性を深く理解するための貴重なデータを提供している。

重イオン衝突特有の現象として超周辺衝突 (Ultra Peripheral Collisions, UPC) と呼ばれるものがある。UPC では数 GeV 以下の事象が多く、液体アルゴンカロリメータのトリガーでは 2 GeV から 5 GeV と、陽子・陽子衝突の場合と比較すると非常に低いエネルギーを持つ事象がターゲットになる。検出器に落とされたエネルギーは LATOME ファームウェアと呼ばれる FPGA 上で再構成されるが、Phase I Upgrade で導入された Super Cell 読み出しでは、UPC 由来の低エネルギー事象に対するトリガー効率が低いことが報告されている。これは、検出器に粒子が入射したタイミング (BCID) を正しく同定できていないことに起因する。そこで提案されたのが、検出器の $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の jFEX と呼ばれる領域で Super Cell の信号を足し合わせ、そのエネルギーが前後の BC と比較して最大である点を採用するアルゴリズムである。このアルゴリズムは Peak Finding Algorithm と呼ばれ、その機能を持つ block を Peak Finder と呼ぶ。この Peak Finder を High-Level Synthesis (HLS) と呼ばれる方法で設計し、LATOME ファームウェアの Output Summing (Osum) へ実装・シミュレーションを行うことが本研究の目的である。

本研究ではまず、実データを用いた Peak Finding Algorithm の検証を行った。その結果、UPC の物理で重要になる 2 GeV から 5 GeV の低いエネルギー領域では 92.2% (統計誤差は $\pm 0.774\%$) の jFEX が BCID を正しく同定できることを確認した。次に、Peak Finder の開発を行った。Peak Finder の機能及びそれを検証するためのテストベンチは C++で設計し、シミュレーションでエラーがないことを確認した。その後、HLS によって処理の並列化などの Architecture を設計し、RTL 生成を行った。最終的な RTL シミュレーションでは、設計した Peak Finder がエラーなく期待されるロジックで動作することを確認した。Peak Finder が単独で問題なく動作することを確認した後は Osum 及び LATOME への統合を段階的に行った。ただし、この LATOME は Layer 1 と呼ばれるもので Osum を含む一部の機能しか実装されておらず完全な統合ではないことに留意したい。統合自体は成功し、HLS によって RTL 生成も行った。このとき、リソース推定使用量が約 1.5% 増加することを確認した。また、RTL シミュレーションでは Peak Finder のロジックが正しいことも確認できた。

本研究によって、最もシンプルな Peak Finder が完成し、Osum への試験的実装が完了した。これにより、より複雑な場合にも対応可能なアルゴリズムの考案や本格的な実装に向けた検証を進める基盤が整った。

第1章　序論

1.1　素粒子物理学の標準模型

素粒子物理学は宇宙を構成する基本的な粒子とそれらの相互作用を解明する学問であり、図 1.1 に示すように、現在までに 17 種類の素粒子が実験的に発見されている。この素粒子物理学の理論的枠組みを支えるのが標準模型 (Standard Model, SM) である。素粒子には物質を構成するクォークとレプトン (それぞれ 6 種類)、それらの間で相互作用を媒介するゲージ粒子がある。ゲージ粒子には電磁相互作用を担う光子、弱い相互作用を担う W ボソンと Z ボソン、強い相互作用を担うグルーオン、質量の起源を説明するために導入されたヒッグス粒子がある。なかでも、ヒッグス粒子は他の素粒子に質量を与えるメカニズムを与える重要な素粒子であり、2012 年にスイスとフランスの国境に設置されている欧州原子核研究機構 (CERN) の Large Hadron Collider (LHC) で、ATLAS 実験と CMS 実験によって発見された [1]。この発見は翌年のノーベル物理学賞受賞に繋がった。

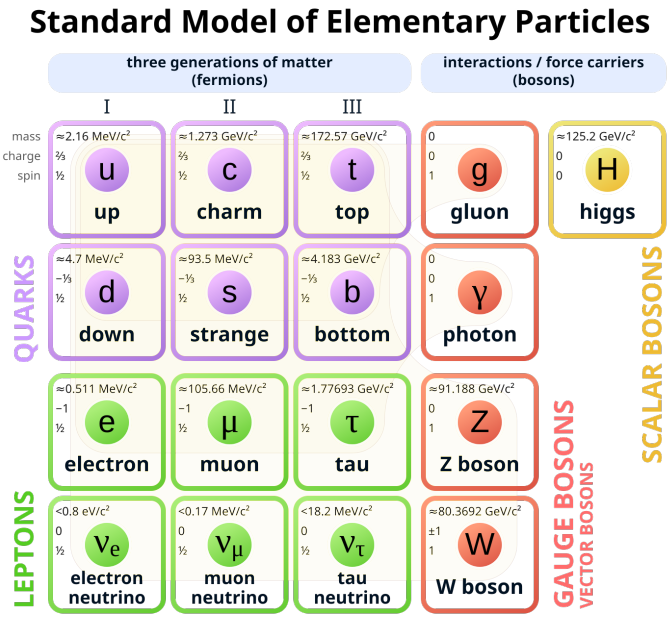


図 1.1: 標準模型で扱われる素粒子 [2]

標準模型は 1970 年代以降多くの実験結果と一致しており、物理学における極めて成功した理論とされている。しかし、いくつかの未解決問題も存在する。例えば、重力が標準模型に含まれていない点、暗黒物質 (Dark Matter, DM) や暗黒エネルギー (Dark Energy, DE) の起源を説明できない点、そしてニュートリノ質量のメカニズムに関する問題などである。これらの点は標準模型を超える新たな物理 (Beyond the Standard Model, BSM) の必要性を示唆している。標準模型が抱えるこれらの問題を

解決する理論として、現在最も有力とされているものの一つに超対称性理論 (SUperSYmmetry theory, SUSY) がある。超対称性理論では、標準模型の各粒子に対応する「超対称性粒子」の存在が予言されており、暗黒物質の候補になる粒子も存在する。CERN に設置された大型ハドロン衝突型加速器 (Large Hadron Collider, LHC) では、標準模型の精度検証に加え、これら超対称性粒子の痕跡を探る重要な実験が行われている。特に私が参加する ATLAS 実験では、光速の 99.999999%まで加速させた高エネルギーの陽子同士を衝突させることで、衝突点から生成される未知の粒子や標準模型を超える物理現象を探索している。一方、宇宙初期に存在したとされる Quark-Gluon Plasma (QGP) の性質や第 3 章で述べる Ultra Peripheral Collisions (UPC) の物理の理解などを目的とした重イオン衝突実験も実施されている。例えば、UPC の光子・光子散乱 ($\gamma\gamma \to \gamma\gamma$) の精密測定は暗黒物質の候補として予言されている Axion-Like Particles (ALPs) の探索に繋がるとされている。また、同じく UPC の $\gamma\gamma \to \tau^+\tau^-$ の精密測定はタウレプトンの異常磁気双極子モーメントに制約を課す [3]。

標準模型の成功と課題、それを克服するための LHC や ATLAS 実験の取り組みは現代物理学における核心的なテーマであり、初期宇宙の謎、余剰次元探索や真空と時空構造の解明などに大きく寄与することが期待されている。

1.2　LHC-ATLAS 実験

LHC は世界最大の重心系エネルギーを誇る周長約 27 km の大型円形粒子加速器であり、標準模型の検証やそれを超える新たな物理現象の探索を目的として、CERN によってスイスとフランスの国境の地下約 100 m に建設された。図 1.2 にその全体像を示す。

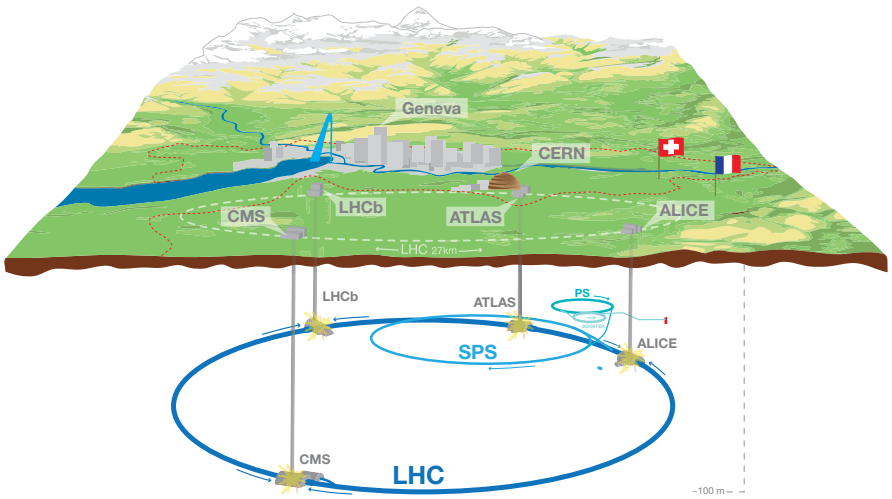


図 1.2: LHC の概要 [4]。ATLAS 検出器は Point 1、ALICE 検出器は Point 2、CMS 検出器は Point 5、LHCb 検出器は Point 8 と呼ばれる位置にある。

LHC では、陽子や重イオン (主に鉛イオン) を光速に近い速度まで加速し、重心系エネルギー 13.6 TeV (重イオンの場合は 5.4 TeV) という非常に高いエネルギーで衝突させることで、宇宙誕生の 1 兆分の 1 秒後の世界を人工的に再現する。衝突点は主に 4 箇所あり、それぞれに ATLAS (A Toroidal LHC ApparatuS), CMS (Compact Muon Solenoid), ALICE (A Large Ion Collider Experiment), LHCb (Large Hadron Collider beauty) と呼ばれる検出器が存在する。

LHC-ATLAS 実験はその名の通り ATLAS 検出器を用いた実験で、主に陽子・陽子衝突によって生成される粒子の精密測定や超対称性粒子の探索等を目的とした国際共同研究プロジェクトである。LHC リング内では約 10^{11} 個の陽子がひと束 (バンチ構造と呼ぶ) となり、衝突点において直径 20 μm ほどの大きさに絞り込まれ、40 MHz (25 ns) の頻度でバンチ交差 (Bunch Crossing, BC) が繰り返される。LHC の設計上、一周期あたり 2,808 のバンチを詰めることができるが、実験時には衝突点での衝突効率や検出器でのデータ取得を最適化するため、使用するバンチ数は 2,808 より少なくなる場合がある[1]。そのため、いくつかのバンチが衝突するといくつかのバンチ分だけ空きができる。これをトレイン構造と呼ぶ。また、各バンチには BCID (Bunch Crossing ID) と呼ばれる識別番号が割り当てられる。BCID は各検出器がセンサーの応答と由来する BC を対応付けるために使われ、同じ BCID を持つデータが事象として再構成される。

LHC-ATLAS 実験における陽子・陽子の非弾性散乱断面積は O(100) mb であり、発生するイベント数は膨大である。その中でヒッグス粒子などの興味のあるイベントの割合は極めて小さい。例えば、重心系エネルギー 13.6 TeV の場合、ヒッグス粒子の生成断面積は約 55 pb であり、10^9 回に 1 回程度（もしくはそれ以下）しか生成されない。超対称性粒子など新物理に関連する素粒子の生成頻度はさらに小さい。限られた計算資源を効率的に活用するためには、興味のある事象のみを選択することが重要であり、背景事象は可能な限り排除することが求められる。そのため、実際の検出器においては興味のあるイベントだけを瞬時に判断する「トリガー」と呼ばれる選択機構が存在する。本研究は重イオン衝突向けのトリガーに関するものである。

[1] 参考までに、本研究で用いた Run 3 の陽子・陽子衝突データのバンチ数は 2,340 である。

1.3　LHC Upgrade 計画とトリガー発行のためのファームウェア開発

1.3.1　LHC Upgrade 計画

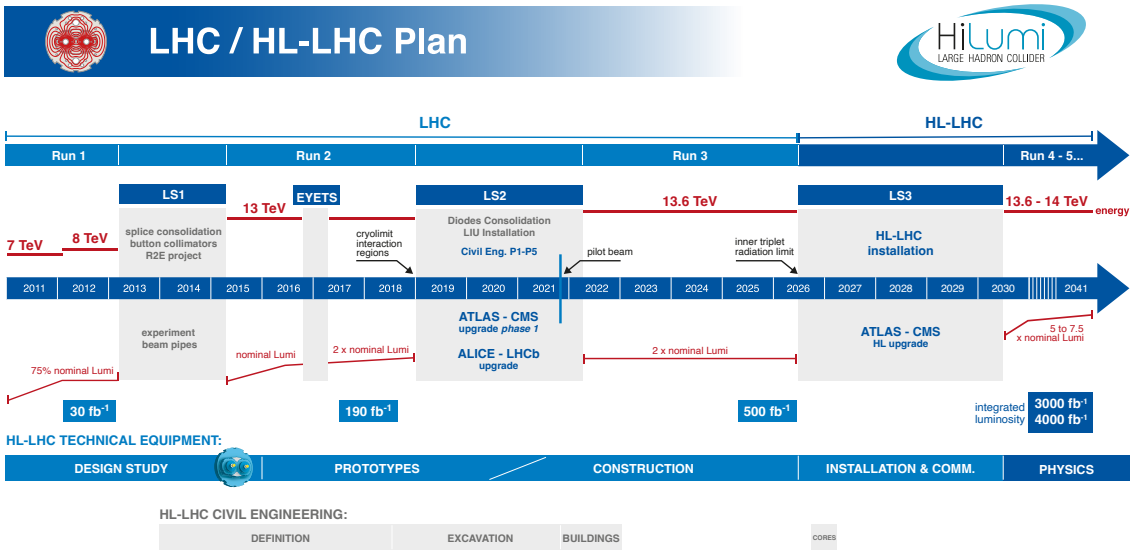
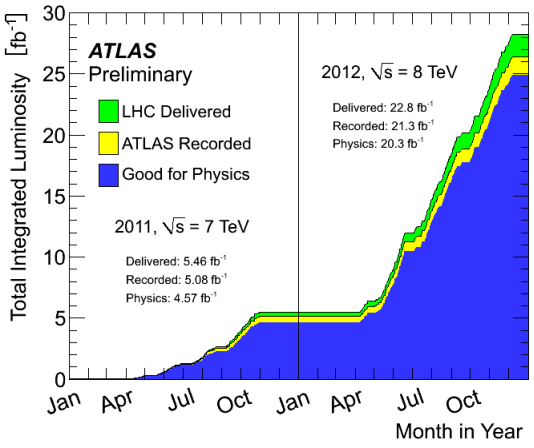
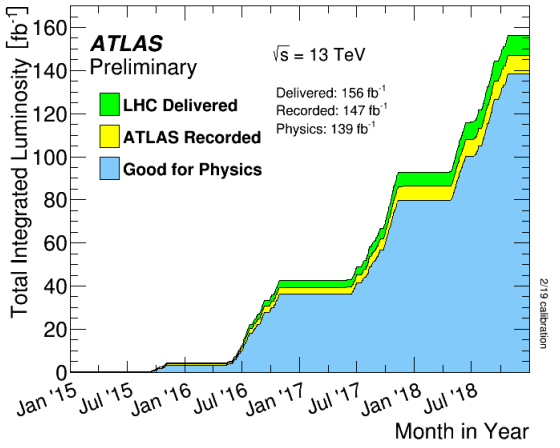


図 1.3: 2024 年 11 月時点での LHC Upgrade 計画のスケジュール [5]。現在は Run 3 期間中で、2026 年から High Luminosity LHC (HL-LHC) に向けた検出器のアップグレード期間 LS3 に入る。

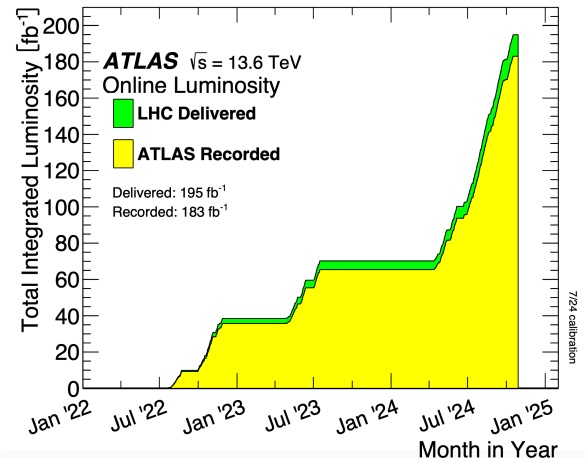
LHC は衝突エネルギーとルミノシティの増強を目指し、段階的にビームインジェクタや検出器の改良を行ってきた (図 1.3)。Run 1 は 2010 年から 2012 年まで行われ、重心系エネルギー 8 TeV を達成、積分ルミノシティ 25 fb^{-1} ほどの物理解析に有用なデータを得ることに成功 (図 1.4a)、ヒッグス粒子の発見に繋がった。その後 LS1 と呼ばれる運転停止期間に入り、衝突エネルギーを上げるため加速器全体の改良などが行われた。2015 年から 2018 年までの Run 2 では重心系エネルギー 13 TeV を達成、物理解析に有用な積分ルミノシティ 139 fb^{-1} を得た (図 1.4b)。Run 2 ではヒッグス粒子の性質についてより詳細に渡り明らかとなり、その実験結果は標準模型が予測するものと高い精度で一致した。その後 LS2 に入り、ミューオン検出器の Small Wheel (SW) の交換や液体アルゴン (Liquid Argon, LAr) カロリメータ検出器のトリガー読み出しエレクトロニクスの改良などが行われた。そして、2024 年現在は Run 3 期間中であり、ヒッグス粒子とその他の粒子との結合定数の精密測定などを目指しデータ取得が行われている。2022 年 7 月から 2025 年 1 月までの ATLAS Recorded ルミノシティ (ATLAS 検出器で記録されたデータ量に相当) は 183 fb^{-1} である (図 1.4c)。



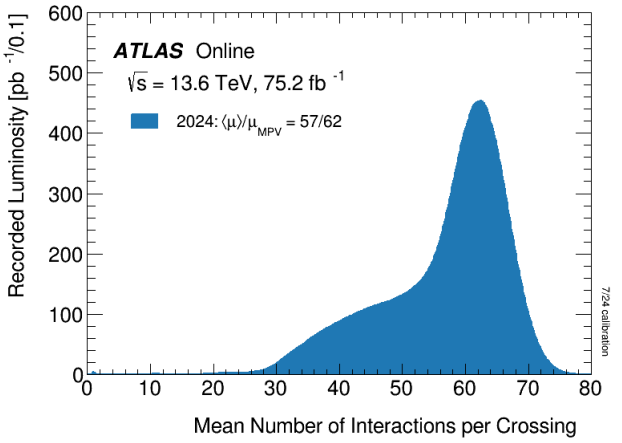
(a) Run 1 の積分ルミノシティ



(b) Run 2 の積分ルミノシティ



(c) Run 3 の積分ルミノシティ



(d) Run 3 (2024 年のみ、8 月 19 日までのデータ) の平均相互作用数の分布。平均相互作用数は一回の BC (陽子・陽子衝突) で起こる非弾性散乱事象の頻度を意味する。2024 年では約 60 である。

図 1.4: 各 Run の積分ルミノシティと 2024 年の平均相互作用数 [6]

Run 3 の後は更なる高輝度化を目指した HL-LHC (High Luminosity LHC) に向けた準備のための期間 LS3 に入る。HL-LHC は 3,000 fb^{-1} から 4,000 fb^{-1} の積分ルミノシティを目指しており、重心系エネルギーは 13.6 TeV から 14 TeV、瞬間ルミノシティは 5 - 7.5×10^{34} $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$、平均相互作用数は 150 から 200 程度になる。平均相互作用数が増えるとトリガーにおける背景事象も増加し、これがフェイク事象[2]として判断される可能性がある。このようなフェイク事象の増加はトリガーレートを圧迫させる。ゆえに、興味のある事象だけを効率良くかつ高速に選択する機構、すなわちトリガーの開発が重要である。次項では現行の ATLAS のトリガーとデータ取得システムについて説明する。

1.3.2 ATLAS のトリガーとデータ取得システム

LHC は 40 MHz の頻度で陽子バンチを交差させるが、限られた計算資源で全ての事象をストレージに記録することは現実的ではない。平均相互作用数 μ が 60 程度だとすると、単位時間当たりに起こる陽子同士の衝突回数はおおよそ $60 \times 40\,\mathrm{MHz} = 2.4 \times 10^{9}\ \mathrm{s}^{-1}$ となるが、興味のある事象が起こる回数

[2]例えば、ヒッグス粒子、W 粒子、Z 粒子、さらには SUSY 粒子のように見える事象が増加する。

はこれよりはるかに小さい。例として、瞬間ルミノシティを $2 \times 10^{34}\,\mathrm{cm^{-2}\,s^{-1}}$、ヒッグス粒子の生成断面積を 50 pb ($5.0 \times 10^{-35}\,\mathrm{cm^2}$) 程度だとすると、単位時間当たりにヒッグス粒子が生成されるレートは $(2 \times 10^{34}) \times (5.0 \times 10^{-35}) = 1\,\mathrm{s^{-1}} = 1\,\mathrm{Hz}$ のように見積もられる。つまり、ヒッグス粒子の生成頻度は 1 秒間に約 1 回である。超対称性粒子などの生成頻度は一般にさらに小さい。

ATLAS 検出器はこの膨大な衝突データの中から興味のある事象を正確かつ高速に選択するトリガーシステムを備えている (図 1.5)。トリガーは 2 段階で行われ、前段は Level-1 (L1) Trigger、後段は High Level Trigger (HLT) と呼ばれている。L1 トリガーではミューオン検出器とカロリメータ検出器の情報を用いて 2.5 μs 以内にハードウェアベースで粗く事象選別を行う。L1 トリガーレートは 100 kHz である。L1 トリガーで許可されたイベントには L1 Accept 信号が発行され、対応するバンチ交差のデータが読み出される。同時に、L1 でトリガーされたオブジェクトの情報は Regions Of Interest (ROI) として定義され HLT に送られる。HLT は L1 トリガーを通過したイベントに対して内部飛跡検出器の情報を追加する。その後、ソフトウェアベースで事象再構成と選別を行い、興味のあるイベントを同定する。このプロセスによりさらにデータ量が削減され、3 kHz のレートでイベント選別が実現される。HLT では 550 ms 以内にデータが処理される。HLT によってイベントが受け入れられると Accept 信号が発行され、データは物理解析のための大容量ストレージディスクに保存される。その後、CERN の Tier-0 と呼ばれるデータセンターに転送され物理解析に使われる。

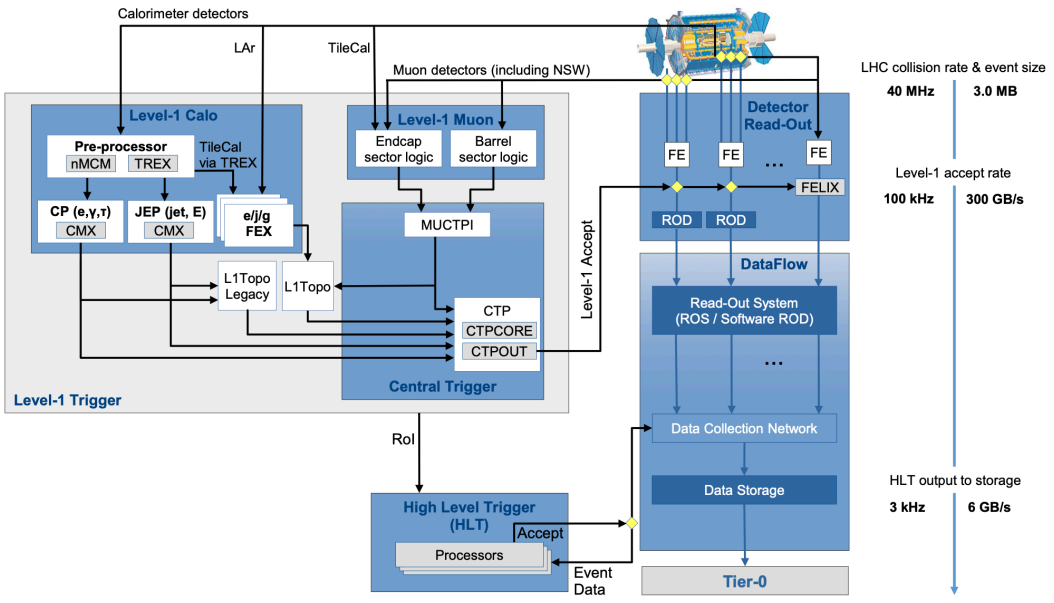
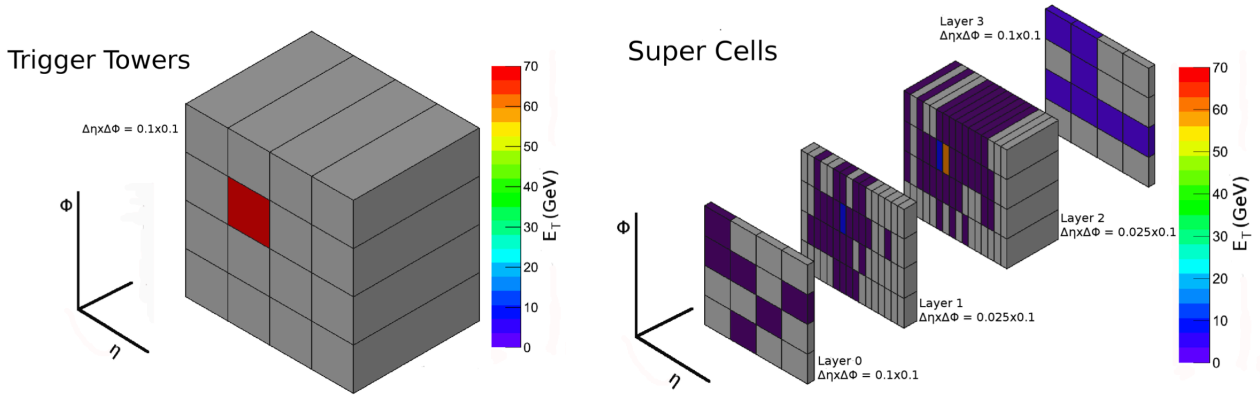


図 1.5: ATLAS 検出器のトリガー・データ取得システム [7]

1.3.3　LAr Calorimeter Phase I Upgrade

本研究は液体アルゴン (Liquid Argon, LAr) カロリメータ検出器のトリガーに関するものなので、ここで Phase I Upgrade で改良された点を簡潔に述べておく。LHC では新物理の探索に向けて年々ルミノシティが向上しているが、ルミノシティが増えると陽子・陽子非弾性散乱事象も増える。事象数が増えれ

ばトリガーにおける背景事象も増えるので、トリガーレートが増加する。したがって、L1 トリガーの要求値を満たすようにトリガーレートを制御することが重要である。液体アルゴンカロリメータの場合は電子・光子 (EM オブジェクト) が主な対象であり、そのトリガーレートは L1Calo (L1 トリガーの一部) が要求する 20 kHz である。このため、2019 年から 2022 年までのアップグレード期間 Phase I Upgrade では、液体アルゴンカロリメータのトリガー用読み出しシステムが刷新された。液体アルゴンカロリメータは主に電磁カロリメータであるため電子・光子のエネルギーを測定する役割を持つ。この場合、一般的な背景事象はハドロン粒子に伴うエネルギーを電子・光子によるものと誤認することによる Background である。Run 2 までの液体アルゴンカロリメータでは $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の Trigger Tower (TT) と呼ばれる領域でエネルギーを足し合わせ、このエネルギーに閾値をかけてトリガーをかけていた (図 1.6a)。Phase I Upgrade ではこのトリガー読み出し単位を 10 倍に細かくした Super Cell (SC) が導入された。トリガーでは SC ごとにエネルギーを読み出すことで、より詳細な電磁シャワー形状解析が可能になり[3]、ハドロン由来の背景事象の除去レートを高め、エネルギー閾値を上げることなく L1Calo の要求値を満たすことに成功した (図 1.6b)。Super Cell による読み出し単位は 4 層構造であり、第 0 層と第 3 層は $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$、第 1 層と第 2 層は $\Delta\eta \times \Delta\phi = 0.025 \times 0.1$ 程度の領域をカバーする。Super Cell の導入に伴いトリガー用読み出しエレクトロニクスの改良も行われたが、これについては次章で詳しく説明する。



(a) Run 2 までのトリガー読み出し：Trigger Tower (TT)　　**(b)** Run 3 以降のトリガー読み出し：Super Cell (SC)

図 1.6: Run 2 と Run 3 のトリガー読み出し [8]。(a) は 16 個の TT で、(b) はそれに対応した 160 個の SC を表す。1 つの TT は典型的には 10 個の SC から構成される。図ではそれぞれ $E_{\mathrm{T}} = 70$ GeV の電子が入射した場合の応答を示しており、セルの細分化によりシャワーの形状がより鮮明になっているのが分かる。

1.3.4 FPGA によるファームウェア設計：現在と将来

Phase I Upgrade 後、液体アルゴンカロリメータで得られた信号は Front-End Board と呼ばれるエレクトロニクスで 25 ns ごとにデジタイズされた後、ATLAS 検出器から 100 m ほど離れた Back-End Board と呼ばれるエレクトロニクスで Super Cell ごとにエネルギー再構成が行われる。このエネルギー再構成はトリガーに関わる重要なパートであり、Back-End Board に搭載されている LATOME Board 上の FPGA (Field Programmable Gate Array) で 15 BC (375 ns, 1 BC=25 ns) のレイテンシ (処理

[3]電子・光子由来の電磁シャワーの半径は $\Delta R \approx 0.08$ で、ハドロンジェットの半径は $\Delta R \approx 0.8$ 程度である。したがって、細かいセル分割を行うことでシャワー半径をより正確に測定でき、トリガー判定の情報として利用することが可能になる。

にかかる時間) で実行される。FPGA はその名の通り、設計者がプログラム可能なロジックエレメント（LUT やフリップフロップなどを含む基本構成単位）を集積したデバイスである。FPGA は専用のツールを用いて動作を設計し、デバイスに書き込むことで論理回路を自由に構成できる。近年はマイコンを内蔵した FPGA も多く存在し、高速並列処理が可能で、リアルタイム演算や特殊な計算に適している。LATOME Board の FPGA 上のファームウェア[4]を LATOME ファームウェアと呼ぶ。この LATOME ファームウェアに書き込まれた内容に基づいて、エネルギー再構成やデータフレームの変更、データの並べ替えなどの処理が行われる。LATOME ファームウェアについては次章で詳しく説明する。

FPGA によるファームウェア設計は、基本的にはハードウェア記述言語 (Hardware Description Language, HDL) を用いて行われる。HDL の代表例としては Verilog や VHDL などがある。実際、Run 3 で使われている LATOME ファームウェアは主に VHDL で記述されており、これは熟練のエンジニアたちがレイテンシを意識しながら正確に動作するよう設計したものである。しかし、HDL によるファームウェア設計にはいくつかの難点がある。例えば、回路動作を詳細に記述しなければならない点や、抽象度が低いためメンテナンスが難しくなる点がそうである。一般に設計が複雑化するほど開発は困難になる。特に、RTL (Register Transfer Level) はクロックやレジスタ間のデータフローを記述する抽象度が低いため、並列処理やタイミング設計の難易度が高いのが課題である。これらの課題を解決するために、現在、HLS (High-Level Synthesis) を用いた LATOME ファームウェアの開発が進められている。HLS は C++などの高級言語で記述された動作記述を入力として RTL を自動生成する技術である。C++などの高級言語は抽象度が高いため、設計期間の短縮やメンテナンスの容易さを実現でき、複雑なアルゴリズムも比較的簡単に実装可能である。また、Architecture の最適化によりレイテンシの短縮とリソースの削減を同時に実現することが可能である。図 1.7 は 2 つの 1 bit の 2 進数に対する加算を、HLS を用いて RTL 生成した例である。専用の HLS ソフトが C++で記述されたコードから自動で HDL を生成するので設計者の負担が減る。また、設計者が実装したい module が正しく動作するか確認するためのテストベンチも C++による設計が可能で、HLS 変換ソフトによりシミュレーションを行うことができる。

[4]ファームウェアはハードウェアを制御するために組み込まれたソフトウェアの一種であり、コンピュータや電子デバイスの基本的な動作を管理し、ハードウェアとソフトウェアの橋渡しのような役割を果たす。

```cpp
// 1bitの2つの正整数の和を求める関数
#include "addition.h"

#pragma hls_design top
void addition(const Num n1, const Num n2, Ans &sum)
{
  sum = n1 + n2;  // 加算の実行
}
```

1 bit　2 bit

(a) 2 つの 1 bit の 2 進数の加算を実現する関数。C++で記述されている。

```
124   always @(posedge clk) begin
125     if ( rst ) begin
126       sum_rsci_idat_0 <= 1'b0;
127     end
128     else if ( ~ (fsm_output[1]) ) begin
129       sum_rsci_idat_0 <= n1_rsci_idat ^ n2_rsci_idat;
130     end
131   end
132   always @(posedge clk) begin
133     if ( rst ) begin
134       sum_rsci_idat_1 <= 1'b0;
135     end
136     else if ( ~ (fsm_output[1]) ) begin
137       sum_rsci_idat_1 <= n1_rsci_idat & n2_rsci_idat;
138     end
139   end
```

sum (LSB)　n1 ⊕ n2　XOR

sum (MSB)　n1 • n2　AND

(b) HLS 変換ソフトにより生成された Verilog のコード (一部)。加算結果 sum の LSB と MSB それぞれが元の入力の排他的論理和 (XOR) と論理積 (AND) で記述されているのが分かる。

n1	n2	sum (MSB)	sum (LSB)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(c) 2 つの 1 bit の 2 進数の加算を実現する真理値表。sum の LSB が XOR、MSB が AND 演算に対応している。

図 1.7: 2 つの 1 bit の 2 進数の加算を実現する C++コードの HLS

1.4 本研究の目的と本論文の構成

ここまで、Phase I Upgrade 以降の液体アルゴンカロリメータの新しいトリガー読み出しと FPGA によるファームウェア設計について述べた。多くの人たちの協力もあり、現行のファームウェアは陽子・陽子衝突実験においては概ね問題なく動作している。

一方、LHC では宇宙初期に存在したとされるクォーク・グルーオンプラズマ (Quark-Gluon Plasma, QGP) 状態の生成とその検証等を目的とした「重イオン衝突実験」も年に 1 ヶ月程度実施されている。衝突に用いられる重イオンは鉛イオンである。ATLAS の液体アルゴンカロリメータではその事象解析のためのトリガー発行が行われているが、2023 年から本格的に運用を開始したデジタルトリガーシステムでは低エネルギーの事象に対して、正しいバンチ交差 (BC) でのトリガー効率が低いことが確認されている。そこで我々は入射粒子が由来する BCID を同定するためにエネルギーが局所最大となる点を採用するアルゴリズム (Peak Finding Algorithm) を 2023 年の実験データ等を用いて検証するとともに、HLS を用いた FPGA ファームウェアへの実装を目的として本研究に取り組んだ。したがって、本論文では重イオン衝突の実データを用いた Peak Finding Algorithm の検証結果及び LATOME ファームウェアの Output Summing (Osum) への HLS による試験的実装とシミュレーション結果について報告する。

本論文では、第 2 章で ATLAS 検出器と液体アルゴンカロリメータ検出器の詳細、そして現行のトリガーシステムについて述べる。第 3 章では重イオン衝突実験向けのトリガー発行アルゴリズム (Peak Finding Algorithm) について述べる。第 4 章では重イオン衝突実験の実データを用いた Peak Finding Algorithm の検証結果について述べる。第 5 章では第 4 章で述べた Peak Finding Algorithm を実装する “Peak Finder” の HLS による開発とシミュレーション結果について述べる。第 6 章では第 5 章で開発した Peak Finder の Osum への統合と検証結果について述べる。第 7 章では結論と今後の展望について述べる。

第2章　ATLAS検出器と液体アルゴンカロリメータ

2.1　ATLAS検出器

ATLAS 検出器は LHC 実験における主要な大型汎用粒子検出器の一つで、標準模型の検証や新物理の探索を目的としている。図 2.1 にその全体像を示す。ATLAS 検出器は全長約 44 m、直径約 25 m、総重量約 7,000 t で、陽子同士の高エネルギー衝突で生成される多種多様な粒子を網羅的に検出するため、複数のサブ検出器とマグネットシステムから構成されている。また、中央の円筒部分をバレル (Barrel)、両端の円面部分をエンドキャップ (End-Cap) と呼ぶ。

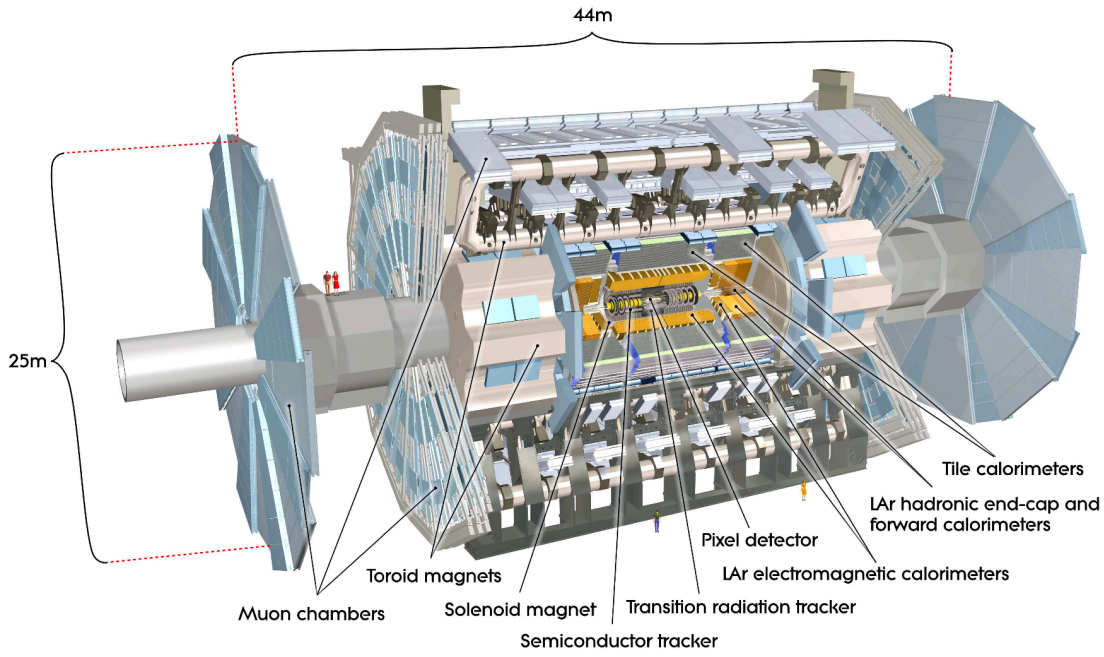


図 2.1: ATLAS 検出器の外観 [9]

サブ検出器は内側から内部飛跡検出器、電磁カロリメータ検出器、ハドロンカロリメータ検出器、ミューオン検出器の 4 種類があり、以下に主要な役割を簡単に示す。

- **内部飛跡検出器**：荷電粒子の飛跡・運動量の測定と衝突点の再構成。
- **電磁カロリメータ**：電磁シャワーを利用して電子や光子のエネルギーを測定。
- **ハドロンカロリメータ**：ハドロンシャワーを利用してハドロンのエネルギーを測定。
- **ミューオン検出器**：ミューオンの飛跡・運動量の測定。

図 2.2 に示すように各サブ検出器はそれぞれ異なる粒子を検出できるようになっている。ニュートリノに限ってはその反応断面積の低さから検出することはできない。しかし、衝突前後の 4 元運動量保存則に基づき、検出器全体の運動量の不一致 (Missing Transverse Energy (MET), E_T^{miss}) を測定することで、その存在と運動量を間接的に推定することが可能である。

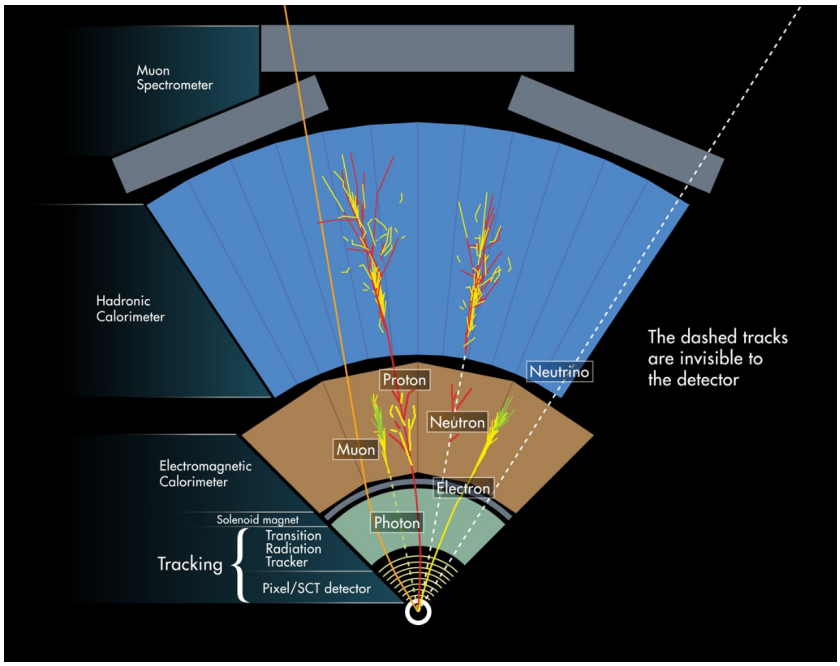


図 2.2: 各サブ検出器で検出可能な粒子 [10]。ATLAS 検出器の断面図を示したものであり、各サブ検出器が同心円状に配置されている。

ATLAS 検出器で検出された粒子の物理量を計算するには座標系が必要である。以下にデカルト座標系と極座標系による定義を示す。どちらも衝突点[1]を原点としている。

[**デカルト座標系**]

- x **軸**：衝突点から LHC リングの中心に向かう方向を正とする。
- y **軸**：衝突点から LHC リング面に垂直、つまり上空に向かう方向を正とする。
- z **軸**：右手系に従って、ビーム軸に沿う方向。

[**極座標系**]

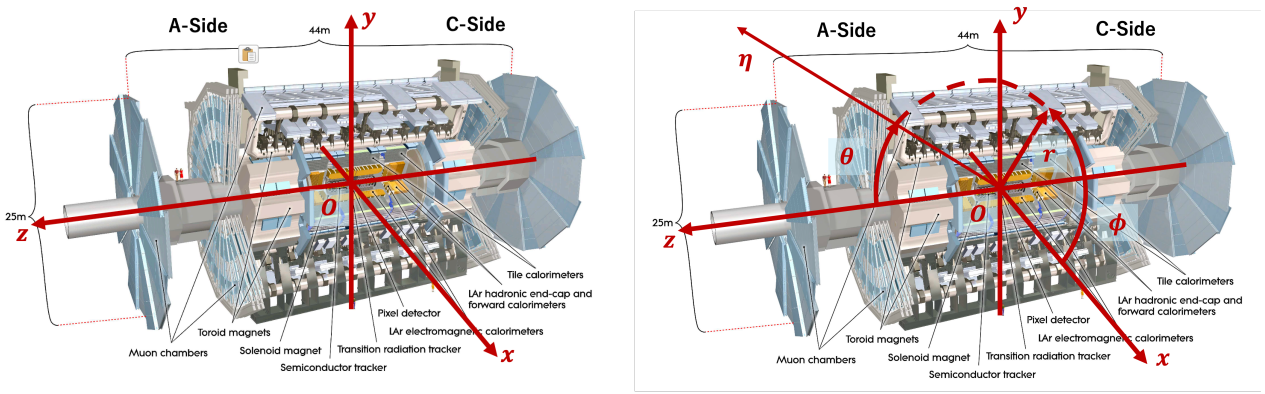
- r：衝突点からの距離を表す。
- θ：z 軸を基準にした粒子の進行方向の角度を表す。
- ϕ：x-y 平面内での角度を示し、粒子の進行方向の方位を表す。

[1]検出器の中心であり、衝突が起こると期待されている点

ハドロンコライダー実験では利便性のため θ の代わりに擬ラピディティ η を用いることが多く、ATLAS 検出器もこれを採用している。以下にその定義を示す。θ の値が 90 度のとき η は 0 で最小値をとる。一方、ビームパイプに沿う方向では $|\eta|$ は大きな値をとる。

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right) \tag{2.1}$$

したがって、極座標表示の場合 θ は η で置き換えて表現することが多い。図 2.3 に ATLAS 検出器の座標系を示す。z 軸の正の方向は A-Side、負の方向は C-Side と呼ばれている。



(a) ATLAS 検出器のデカルト座標系　　**(b)** ATLAS 検出器の極座標系

図 2.3: ATLAS 検出器の座標系

2.1.1　マグネットシステム

ATLAS 検出器のマグネットシステム (図 2.4) は、粒子の運動量を測定するための磁場を生成する重要な構成要素であり、Central Solenoid、Barrel Toroid、および End-cap Toroid の 3 種類が存在する。

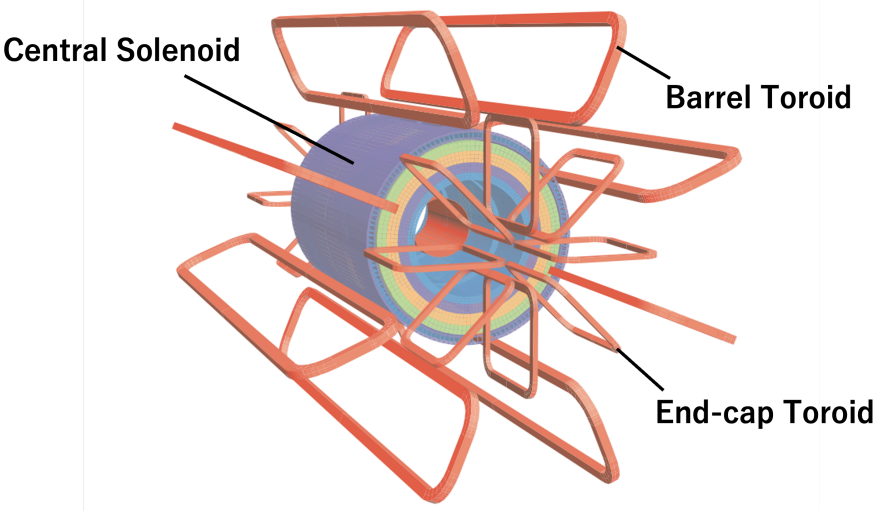


図 2.4: ATLAS 検出器のマグネットシステム [11]

Central Solenoid は内部飛跡検出器のすぐ外側に設置され、同心円状の 2 T のソレノイド磁場を生成することで荷電粒子の進行方向を曲げ、その方向と飛跡から電荷と運動量の測定を可能にしている。Barrel Toroid は検出器の中央部分を取り囲むように配置され、0.5 T のトロイド磁場を発生させる。ミューオン検出器はこの磁場を利用して粒子の運動量測定を行う。End-cap Toroid は Barrel Toroid を補完し、前方領域の粒子測定を担う。その磁場は 1 T である。これらのマグネットシステムにより、全方位における精密な運動量測定が可能になり、ATLAS 検出器の粒子識別能力を向上させている。

2.1.2　内部飛跡検出器 (Inner Detector)

内部飛跡検出器 (Inner Detector, ID) は、衝突点近傍での荷電粒子の軌跡を高精度に測定するため 3 つのサブ検出器、Pixel Detector (PD)、Silicon Strip Tracker (SCT)、Transition Radiation Tracker (TRT) から構成されている。全長は約 6.2m、直径は約 2.1m で円筒構造を持ち、Central Solenoid から 2 T の磁場を受けている。図 2.5 にその全体像を示す。磁場を B [T]、飛跡の曲率半径を ρ [m]、磁場と垂直な荷電粒子の運動量の大きさを $p_\perp$ [GeV] とすると以下の関係が成り立つ。この関係式を用いて、既知の磁場と測定された曲率半径から荷電粒子の電荷と運動量を求めることができる。

$$p_\perp = 0.3B\rho \tag{2.2}$$

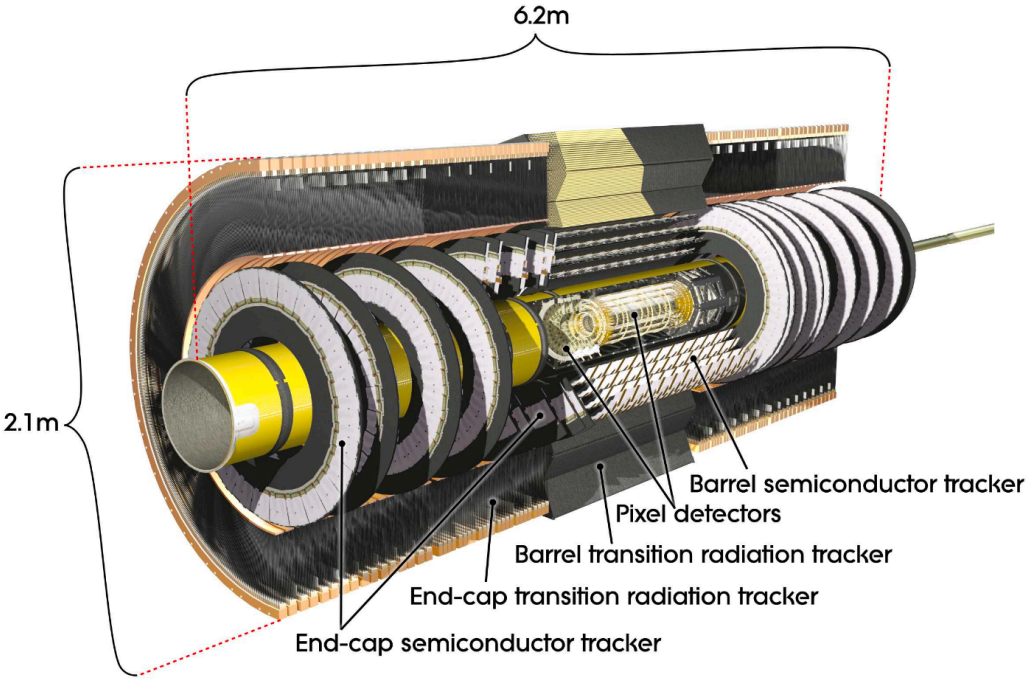


図 2.5: 内部飛跡検出器の全体像 [11]。内側から PD, SCT, TRT の順で構成されている。

2.1.3　カロリメータ (Calorimeter)

ATLAS のカロリメータはサンプリング型であり、電子や光子のエネルギーを測定するための電磁カロリメータとその外側にあるハドロンのエネルギーを測定するためのハドロンカロリメータから構成されている。図 2.6 にその全体像を示す。カロリメータに入射した電子や光子、ハドロンは物質との相互作用によりカスケードシャワーを起こし、エネルギーを損出する。そのエネルギーを吸収し、電気信号として取り出すことでエネルギーの測定を行う。

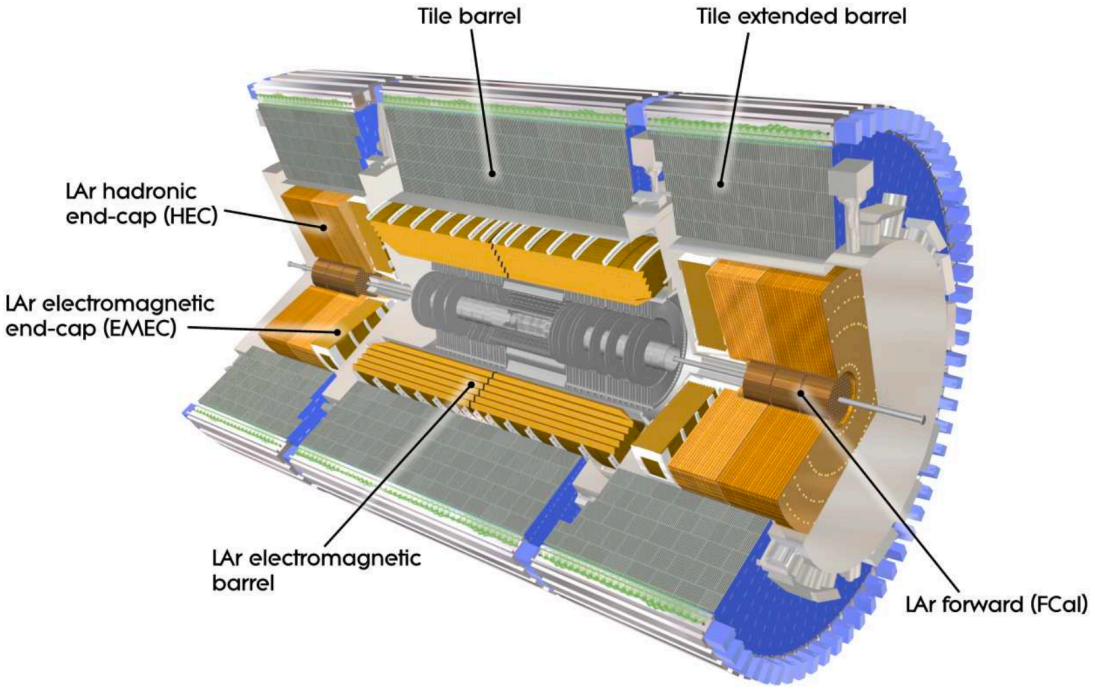


図 2.6: カロリメータ検出器の全体像 [11]。黄色で示されている領域が液体アルゴンカロリメータである。その外側をタイルカロリメータが覆っている。カロリメータ全体では $|\eta| < 4.9$ の領域をカバーしている。

電磁カロリメータ全域、及び Forward と End-Cap 領域に含まれるハドロンカロリメータは検出層に液体アルゴン (Liquid Argon, LAr) が使用されており、これらを総称して液体アルゴン (LAr) カロリメータと呼ぶ。液体アルゴンカロリメータは測定範囲に応じて 4 つのセグメントに分けられており、Barrel 領域をカバーする LAr ElectroMagnetic Barrel (EMB)、End-Cap 領域をカバーする LAr ElectroMagnetic End-Cap (EMEC) と LAr Hadronic End-Cap (HEC)、そして前方領域をカバーする LAr Forward Calorimeter (FCal) である。一方、Barrel 領域のハドロンカロリメータはタイルカロリメータと呼ばれ、検出層にプラスチックシンチレータ、吸収層に鉄が使用され、$|\eta| < 1.7$ の領域をカバーしている。液体アルゴンカロリメータの吸収層は Barrel と一部の End-Cap 領域では鉛、その他の End-Cap 領域では銅、Forward 領域では銅とタングステンが使われている。

表 2.1 及び表 2.2 にそれぞれ電磁・ハドロンカロリメータの構成、カバー範囲、吸収層、検出層をまとめた。FCal に関しては、FCal 1 が電磁カロリメータで FCal 2, 3 がハドロンカロリメータである。

表 2.1: 電磁カロリメータの概要 [12]

	電磁カロリメータ		
名称	EMB	EMEC	FCal 1
$\|\eta\|$ 範囲	0 - 1.475	1.375 - 3.2	3.1 - 4.9
吸収層	鉛	鉛	銅
検出層	液体アルゴン	液体アルゴン	液体アルゴン

表 2.2: ハドロンカロリメータの概要 [12]

	ハドロンカロリメータ		
名称	Tile Barrel	HEC 1, 2	FCal 2, 3
$\|\eta\|$ 範囲	0 - 1.7	1.5 - 3.2	3.2 - 4.9
吸収層	鉄	銅	タングステン
検出層	プラスチックシンチレータ	液体アルゴン	液体アルゴン

2.1.4　ミューオン検出器 (Muon Detector)

ミューオンは電子に比べて約 200 倍質量が大きく制動放射がほとんど起きないため全ての検出器を容易に通過する。したがって、これらミューオンの運動量を測定する目的で、ミューオン検出器は ATLAS 検出器の最も外側に設置されている (図 2.7)。ミューオンの軌道はトロイド磁石によって曲げられるため、その飛跡を再構成することで運動量が求められる。

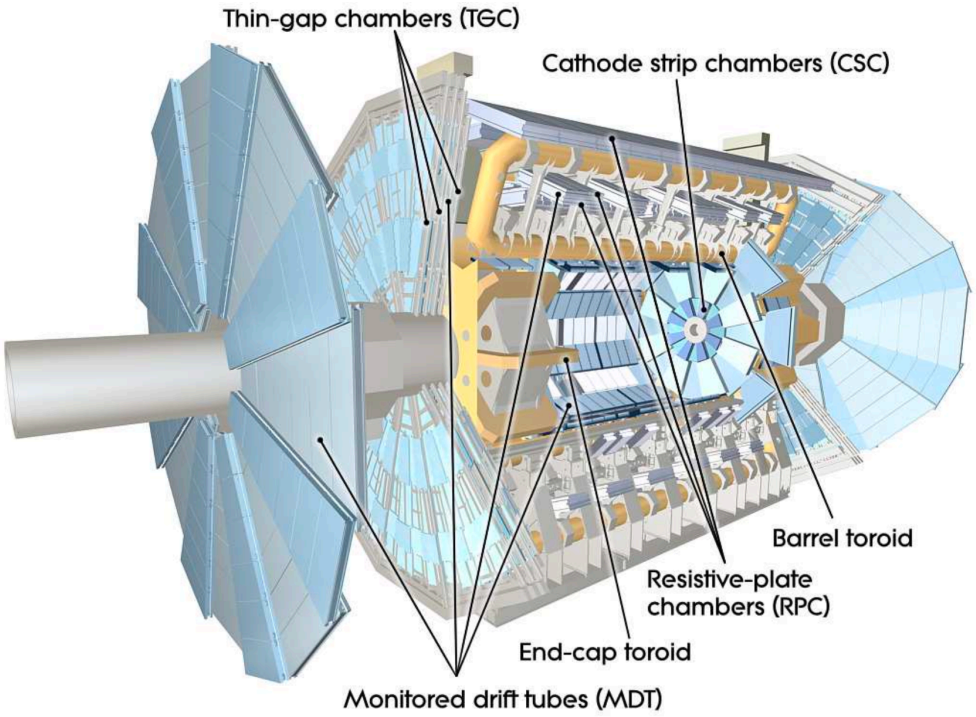


図 2.7: ミューオン検出器の全体像 [11]

ミューオン検出器は Barrel 領域の Resistive-Plate Chambers (RPC)、End-Cap 領域の Cathode Strip Chambers (CSC) と Thin-Gap Chambers (TGC)、Barrel と End-Cap 両領域の Monitored Drift Tubes (MDT) の 4 つのサブ検出器から構成される。中でも、TGC と RPC はトリガーに用いられる。また、Phase I Upgrade により End-Cap Toroid の内側に New Small Wheel (NSW) が導入された。NSW では、Small Strip Thin Gap Chambers (sTGC) と MicroMegas (MM) を組み合わせた新しい技術が採用された。

2.2　液体アルゴンカロリメータ

本研究は液体アルゴンカロリメータのトリガーに関するものなので、ここで液体アルゴンカロリメータについて詳しく述べる。図 2.8 にその全体像を示す。2.1.3 項で述べたように液体アルゴンカロリメータは 4 つのセグメントに分けられており、Barrel 領域の EMB、End-Cap 領域の EMEC、HEC、そして Forward 領域の FCal がある。このうち、EMB と EMEC は図 2.9a に示すようにアコーディオン構造と呼ばれる円周方向に折り重なり合うような構造をとっている。このような構造をとることで ϕ 方向の不感領域をなくしている。

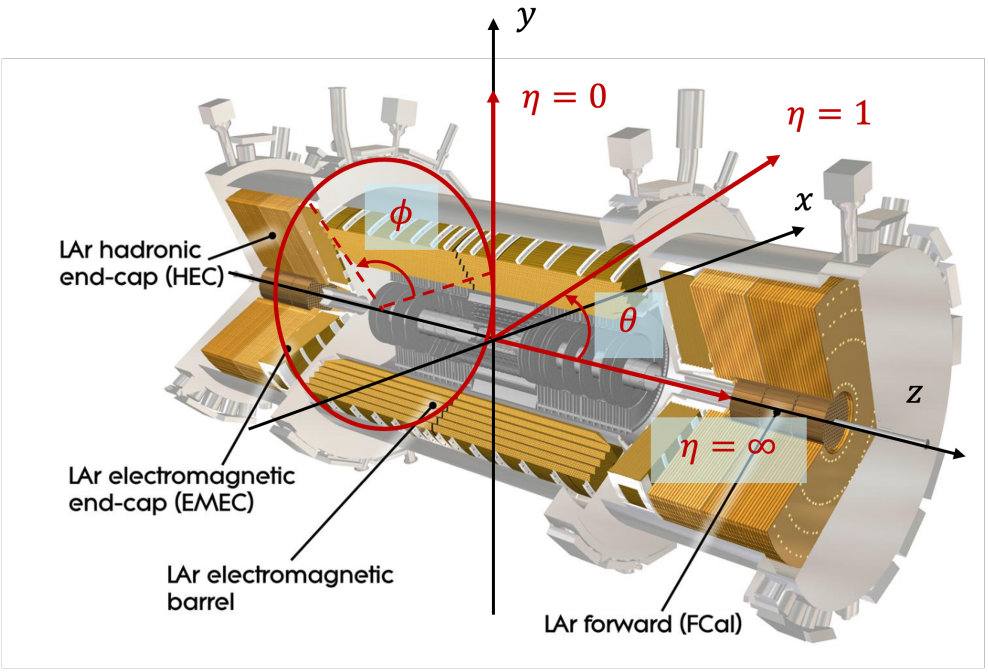
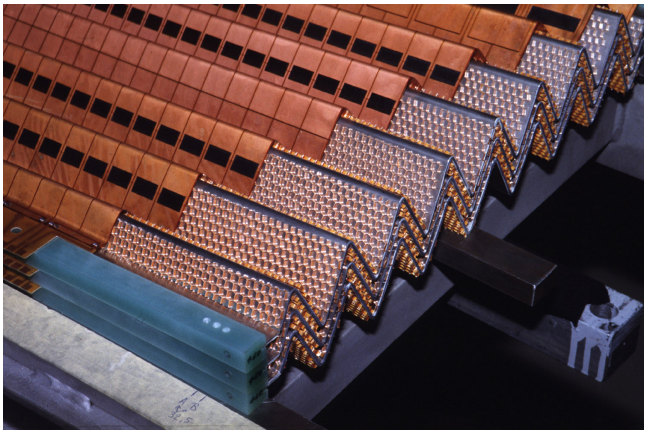


図 2.8: 液体アルゴンカロリメータの全体像 [13]。内部飛跡検出器の外側、ミューオン検出器の内側に位置する。

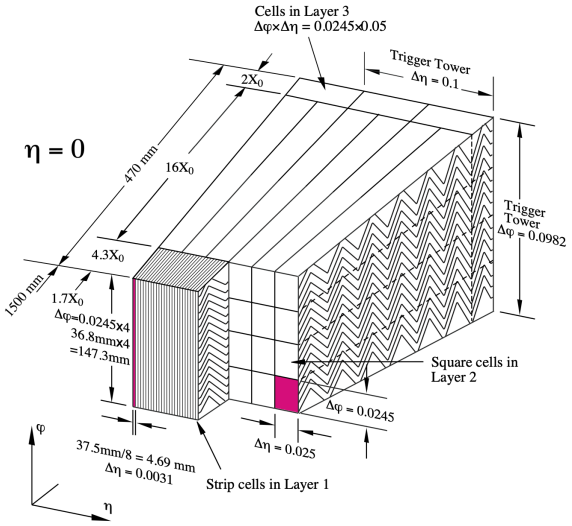
液体アルゴンカロリメータは入射粒子のエネルギーを高精度に測定するために r 方向に 4 層の構造を持つ (図 2.9b)。これらの層は内側から Presampler、Front layer、Middle layer, Back layer と呼ばれており、各 layer の最小読み出し単位 (LAr cell) は $\Delta\eta \times \Delta\phi = 0.025 \times 0.1$、$\Delta\eta \times \Delta\phi = 0.003125 \times 0.1$、$\Delta\eta \times \Delta\phi = 0.025 \times 0.025$、$\Delta\eta \times \Delta\phi = 0.05 \times 0.025$ である[2]。Presampler は $|\eta| < 1.8$ の領域のみに設置され、これは粒子がカロリメータに入射する前に落としたエネルギーを見積もる役割を果たす。また、

[2]Front layer が η 方向に細かくなっているのは、主に中性パイ中間子 (π^0) の崩壊モードが関係している。π^0 は 8.4×10^{-17} 秒という非常に短い寿命で電磁相互作用により 2 つの光子 (γ) に崩壊する。これらの光子はほとんど同方向に飛ぶため、興味のある事象から来る単一の光子と区別するためには細かいセグメントでシャワー形状をより正確に捉える必要がある。

Middle layer は r 方向の奥行きが最も長く、電子や光子はそのエネルギーのほとんどをこの layer に落とす。



(a) 液体アルゴンカロリメータのアコーディオン構造 [14]。液体アルゴンの層と鉛の層が交互に配置されている。



(b) 液体アルゴンカロリメータの 4 層構造 [11]。しかし、この図には Presampler は描かれていない。

図 2.9: 液体アルゴンカロリメータの構造

EMB (ElectroMagnetic Barrel)

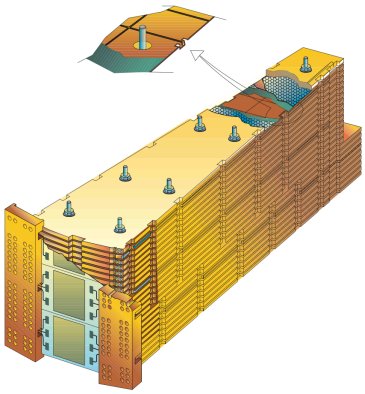
EMB の吸収層は鉛であり、$|\eta| < 1.475$ の領域をカバーしている。z 軸方向の長さは 6.4 m、内半径は 2.8 m、外半径は 4 m、総重量 114 t である。全領域でアコーディオン構造を持つ。

EMEC (ElectroMagnetic End-Cap)

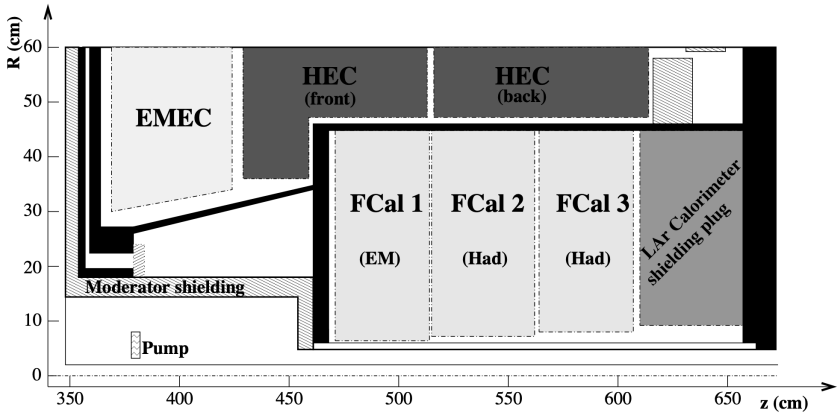
EMEC は EMB の両側に Wheel 形状で設置され、吸収層は鉛で、$1.375 < |\eta| < 3.2$ の領域をカバーしている。それぞれの Wheel の厚さは 63 cm、重量は 27 t である。また、Wheel の内半径は 330 mm、外半径は 2098 mm である。

HEC (Hadronic End-Cap)

HEC は図 2.10a に示すように平板状のモジュールを 32 枚重ね合わせた形状を持つハドロンカロリメータであり、吸収層に銅を採用し、$1.5 < |\eta| < 3.2$ の領域をカバーしている。また HEC は衝突点に近い方から HEC 1 (Front) と HEC 2 (Back) に分けられる (図 2.10b)。



(a) HEC の構造。



(b) 液体アルゴンカロリメータの End-Cap 構造。左側が衝突点の方向である。

図 2.10: HEC と End-Cap の構造 [11]

FCal (Forward Calorimeter)

FCal は他のセグメントと異なり、電磁カロリメータとハドロンカロリメータが混在する。図 2.10b に示されるように z 軸方向に 3 層構造 (FCal 1, FCal 2, FCal 3) を持ち、FCal 1 は吸収層が銅の電磁カロリメータ、FCal 2 と FCal 3 は吸収層がタングステンのハドロンカロリメータである。カバー範囲は $3.1 < |\eta| < 4.9$ でビーム軸に平行なストロー構造を持つ。

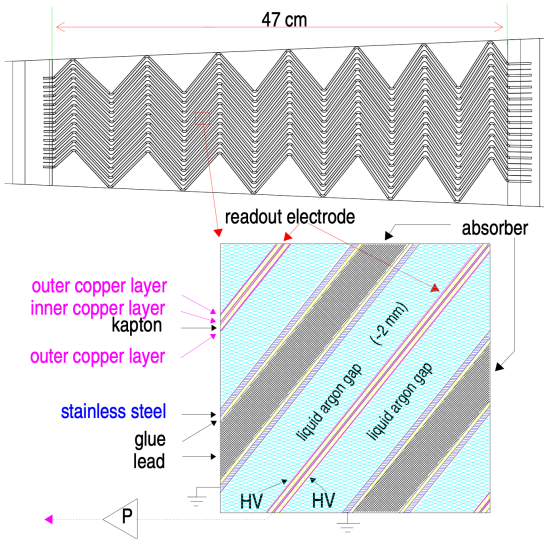
2.3　デジタルトリガーによる信号検出

2.3.1　液体アルゴン検出器からの信号

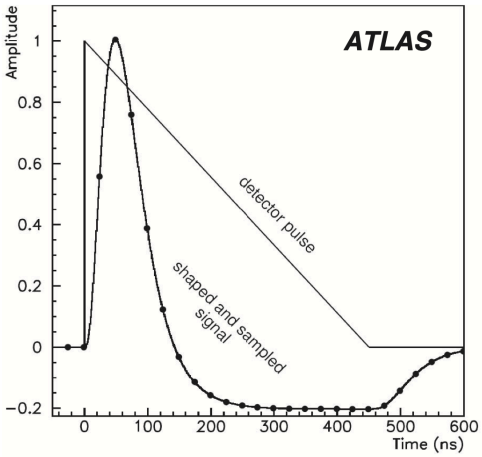
衝突点から生成された非常に高いエネルギーを持つ電子や光子が液体アルゴンカロリメータに入射すると吸収層で電磁シャワーが発生する[3]。この電磁シャワーで生成された荷電粒子は厚さ 2 mm 程度の液体アルゴンの層に到達するとアルゴン原子を電離させる (図 2.11a)。その後、電離電子はこのギャップ間に印加された 2 kV の電圧により銅電極に収集され、最終的に信号が電圧の変化として読み出される。この信号は図 2.11b にあるような三角波と呼ばれるもので、その波高は入射粒子のエネルギーと線形の関係を持つ。三角波はそのままだと扱いにくいのでシェイパーでバイポーラ波形に変換される。この変換によって入射粒子のエネルギーと波高の線形性は失われない。バイポーラ波形は約 600 ns の長さを持ち、時間積分を実行するとその値がゼロになる性質がある。これにより、低い波高を持つパイルアップ[4]が複数来たときにバイポーラ波形の正負が相殺され Baseline (信号の基準点) が一定に保たれる。後段の読み出しシステムではこのバイポーラ波形の波高をフィルタリングアルゴリズムを用いて計算し、入射粒子が由来する BC の同定とエネルギーの再構成を行う。

[3] 吸収層では電子は制動放射、光子は電子陽電子対生成によりエネルギーを失い、次々と高次粒子が生成される。この過程は臨界エネルギーに達するまで繰り返され、最終的に入射粒子のエネルギーの一部を受け取る形で多くの荷電粒子が生成される。

[4] 1 回のバンチ交差で複数の衝突事象が同時に発生する現象。



(a) 液体アルゴンカロリメータの検出原理 [15]。上は z 軸方向から見た断面図で、左側が衝突点である。下はその拡大図で、鉛と液体アルゴンが交互に並べられ、電磁シャワーを繰り返し誘発し、電子や光子のエネルギーを完全に吸収する。中央のピンクの線が 3 層の銅電極である。



(b) 液体アルゴンカロリメータから得られる三角波とバイポーラ波形 [8]。波形は規格化されている。バイポーラ波形は 25 ns ごとにサンプリングされ、後段の信号処理に使われる。

図 2.11: 液体アルゴンカロリメータの信号検出原理と三角波

2.3.2　Super Cell

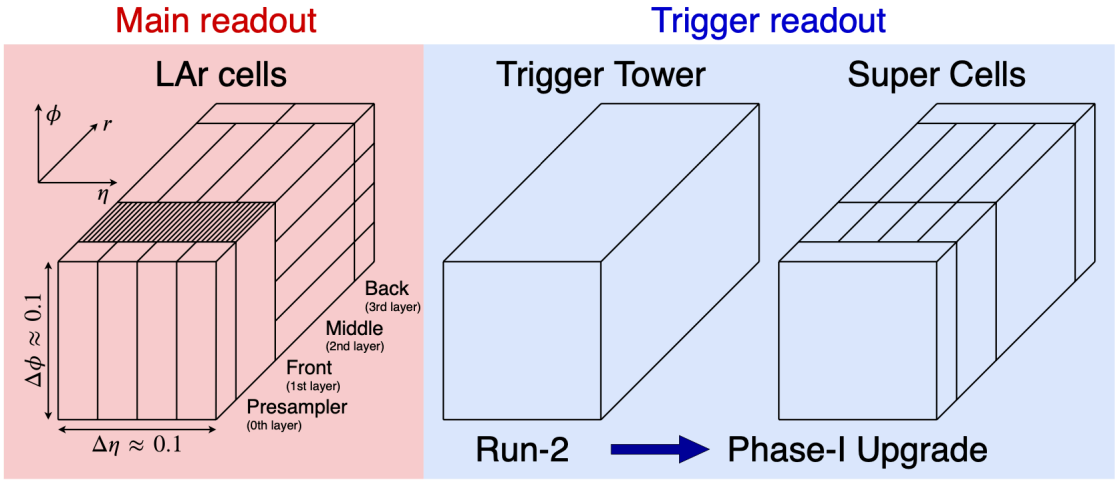


図 2.12: メイン読み出しセルとトリガー読み出しセル [16]。Super Cell の導入により、エネルギー閾値を上げることなくトリガーレートを保つことが可能になった。図は Barrel 領域の TT と SC の構造を示しており、典型的には 1 TT = 10 SC である。しかし、検出器の領域によっては SC が 1 層または 2 層しかない場合もある。これは検出器の構造に依存している。

液体アルゴンカロリメータからの信号は検出器を構成する LAr cell と呼ばれる最小単位ごとに読み出される (図 2.12 左)。LAr cell の数は液体アルゴンカロリメータ全体で 182,418 あり、初段トリガーのためにリアルタイムで毎 BC 全てのチャンネルから精密にエネルギー計算を実行してトリガーをかけることは時間の制約と計算資源の観点から不可能である。そのため、液体アルゴンカロリメータでは検出器のある領域 ($\Delta\eta \times \Delta\phi$) ごとに LAr cell の信号をアナログで足し合わせ、足し合わされたエネルギーを

用いてトリガーのためのリアルタイム再構成を行う。この一連の処理を行うエレクトロニクス上のパスをトリガー読み出しという。一方、初段トリガーで選び出された事象に対しては LAr cell ごとに信号のデジタル処理が行われ、より詳細なエネルギー計算が実行される。これもエレクトロニクス上で行われ、そのパスをメイン読み出しという。

1.3.1 項では LHC のルミノシティの増加がトリガーレートに影響を及ぼすことについて述べた。液体アルゴンカロリメータでは LAr cell で得られたバイポーラ波形を検出器のある領域で足し合わせて、トリガー判断の条件に用いる。Run 2 までは $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の領域で Presampler から Back layer まで信号を足し合わせた Trigger Tower (TT) をトリガー読み出し最小単位としていた (図 2.12 中央)。しかし、Phase I Upgrade 以降は、より詳細なシャワー形状解析を可能にするため、Trigger Tower を 10 倍にセル分割した Super Cell (SC) がトリガー読み出し最小単位になった (図 2.12 右)。Super Cell は 4 層構造を持ち、セルサイズは Presampler が $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$、Front layer と Middle layer が $\Delta\eta \times \Delta\phi = 0.025 \times 0.1$、Back layer が $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ である。LAr cell とのサイズ比較を表 2.3 にまとめた。

表 2.3: LAr cell, Trigger Tower, Super Cell の分割範囲 [17]

Layer		LAr cell	Trigger Tower		Super Cell	
		$\Delta\eta \times \Delta\phi$	$n_\eta \times n_\phi$	$\Delta\eta \times \Delta\phi$	$n_\eta \times n_\phi$	$\Delta\eta \times \Delta\phi$
0	Presampler	0.025 × 0.1	4 × 1	0.1 × 0.1	4 × 1	0.1 × 0.1
1	Front	0.003125 × 0.1	32 × 1		8 × 1	0.025 × 0.1
2	Middle	0.025 × 0.025	4 × 4		1 × 4	0.025 × 0.1
3	Back	0.05 × 0.025	2 × 4		2 × 4	0.1 × 0.1

2.3.3　デジタルトリガーによる信号検出

図 2.13 は Run 3 以降の液体アルゴンカロリメータの信号読み出しエレクトロニクスの全容である。液体アルゴンカロリメータからの信号は、Front-End エレクトロニクスで SC ごとに信号が足し合わされた後、40 MHz でサンプリングされる。その後、Back-End エレクトロニクスでエネルギー再構成が行われ、Level-1 Calorimeter Trigger System (L1Calo) の Feature Extractor (FEX) と呼ばれるところにトリガー発行のための情報が送られる。以下、順に説明する。

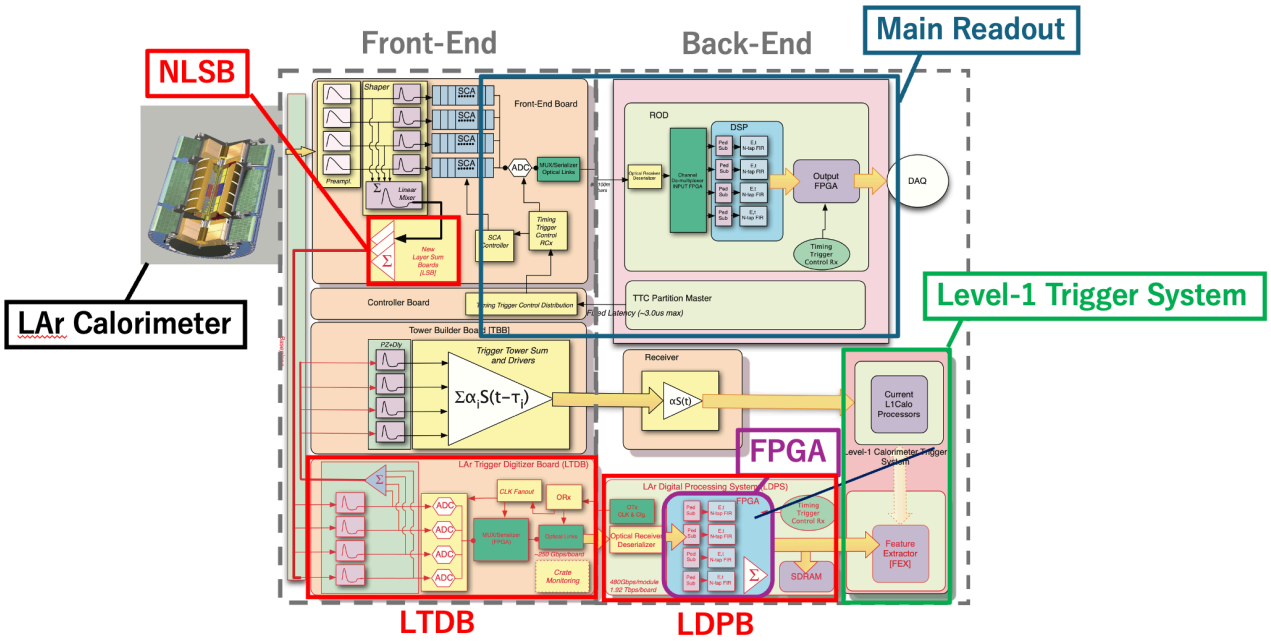


図 2.13: 液体アルゴンカロリメータの信号読み出しエレクトロニクス [8]

Front-End エレクトロニクス

液体アルゴンカロリメータの各 LAr cell で得られた三角波は Barrel と End-Cap の端を取り囲むように配置された Front-End Crate 内の Front-End Board に入る。Front-End Board では、まず Preamplifier によって信号の増幅が行われ、Shaper でバイポーラ波形に変換される。その後、精密なエネルギー計算のためのメイン読み出しパスとトリガーのためのトリガー読み出しパスに分岐する。メイン読み出しパスでは Switched Capacitor Array (SCA) で一時的に信号が保存される。一方、トリガー読み出しパスでは、New Layer Sum Board (NLSB) と呼ばれる plug-in card でアナログレベルでの信号の足し合わせが行われる。この足し合わせは各 SC の領域に対応している。その後、足し合わされた信号は Baseplane を介して LAr Trigger Digitizer Board (LTDB) に送られる。

LTDB は 124 枚存在し、1 枚の LTDB は最大で 320 個の SC を処理できる。LTDB では 40 MHz (25 ns) の頻度でバイポーラ波形のサンプリングが行われる。量子化ビット数は 12 bit であり、波高を 0 から 4095 までの数値で符号化する。1 枚の LTDB は最終的に 40 本の光ファイバーを用いて、100 m ほど離れた Back-End エレクトロニクスの LAr Digital Processing Board (LDPB) に ADC データ (デジタル化された信号) を伝送する。このとき、光ファイバー 1 本あたりの通信速度は 5.12 Gbps であり、1 枚の LTDB では 204.8 Gbps、124 枚全ての LTDB では約 25 Tbps である。

Back-End エレクトロニクス

LDPB は 31 枚存在し、それぞれに Advanced Mezzanine Card (AMC) として実装された 4 枚の LATOME Board が搭載されている[5]。1 枚の LATOME Board は最大で 320 個の SC を処理可能であり、搭載された FPGA でエネルギーの再構成が行われる (図 2.14)。このエネルギー再構成を行う FPGA

[5] すべての LDPB に 4 枚の LATOME Board があるわけではない。LATOME Board の総数は 116 である。

ファームウェアは LAr Trigger Processing Mezzanine (LATOME) ファームウェアと呼ばれ、その詳細については次節で述べる。

1 枚の LATOME Board は 40 本の光ファイバーを通じてエネルギー情報を L1Calo の FEX に送信する。この送信において、光ファイバー 1 本あたりの通信速度は 11.2 Gbps であり、31 枚全ての LDPB では約 40 Tbps である。

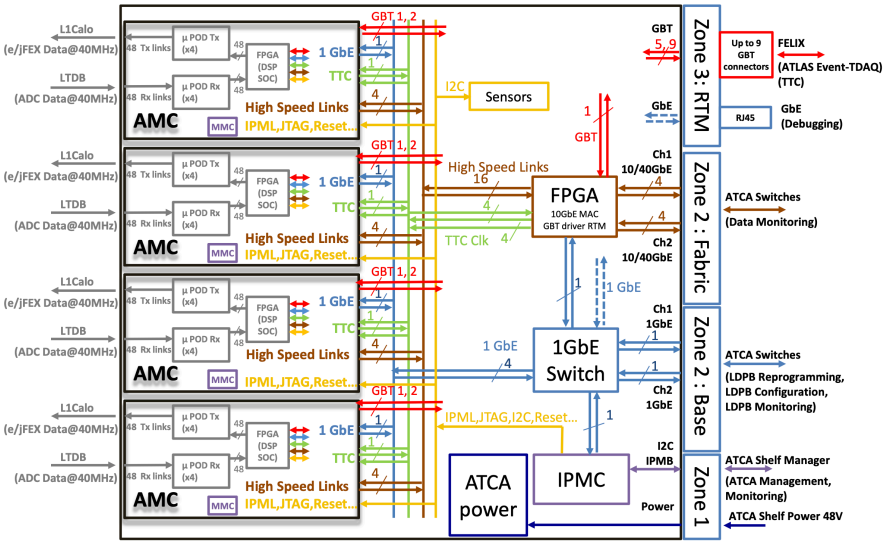


図 2.14: LDPB の概要 [8]。図は 1 枚の LDPB を表している。4 枚の LATOME Board それぞれに搭載された FPGA は LTDB から 40 MHz の頻度で ADC データを受信し、40 MHz の頻度で FEX にエネルギー情報を送信する。

Feature Extractor (FEX)

LATOME ファームウェアで正しい BC で計算された各 SC の横エネルギー E_T は表 2.4 に示される範囲 (eFEX, jFEX, gFEX) で足し合わせれ、40 MHz のクロックに同期させて L1Calo の Feature Extractor (FEX) に送られる[6]。eFEX は電子・光子・タウの同定、jFEX はジェットと消失横運動量 E_T^{miss} の同定、gFEX は半径の大きいジェットの同定を担当する。各オブジェクトの同定にはそれぞれ独自の Cluster-Finding Algorithm が用いられている。

表 2.4: eFEX, jFEX, gFEX の足し合わせ範囲 [18]

	eFEX	jFEX	gFEX
ターゲット	電子/光子/タウ	ジェット	半径の大きいジェット
$\Delta\eta \times \Delta\phi$	0.025×0.1 or 0.1×0.1	0.1×0.1	0.2×0.2
r 方向の粒度	各 layer で独立	全ての layer で合計	全ての layer で合計
総数	34,048	5,760	1,984

[6] eFEX は electron、jFEX は jet、gFEX は global という意味でトリガーのためのオブジェクト識別に必要な情報を含んでいる。

2.4　LATOME ファームウェア

LATOME ファームウェアは液体アルゴンカロリメータに入射した粒子のタイミング同定 (BC の同定) とエネルギー計算を行う、デジタルトリガーにおいて最も重要な部分である。以下にその役割をまとめる。

- 1 つまたは複数の LTDB から 48 本の光ファイバーでデータを受信し、最大で 320 個の SC のデータを処理する。
- デジタルフィルタリングアルゴリズムの適用により、SC の横エネルギー E_T を 25 ns ごとに再構成し、そのエネルギーが落とされた BCID を同定する。また、FEX に 25 ns ごとに計算結果を送信する。
- TDAQ の読み出しチェーンやローカルモニタリングに向け、L1A (Level-1 Accept) やその他のリクエストに応じてデータを処理しバッファする。

図 2.15 に LATOME ファームウェアが実装されている LATOME Board の外観を示す。LATOME Board には 4 つのコネクタがあり、LTDB からの受信に 48 本、FEX への送信に 48 本、計 96 本の光ファイバーが接続されている。実際に意味のあるデータの送受信に使われているものは 40 本ずつである。

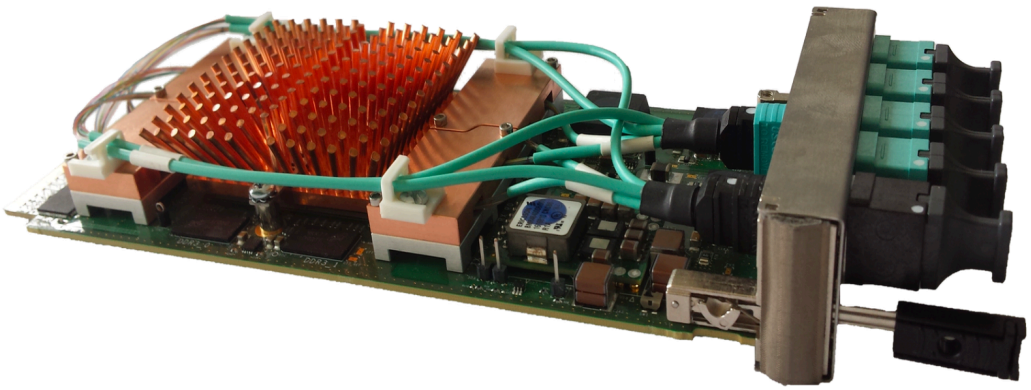


図 2.15: LATOME Board の外観 [18]。銅色のヒートシンクの下に FPGA がある。LATOME Board には 48 本の受信用光ファイバーと 48 本の送信用光ファイバーが MicroPOD を通して接続されている。

1 枚の LATOME Board は約 200 Gbps の受信速度と約 450 Gbps の送信速度を達成する。搭載されている FPGA は Intel 社製の Arria 10 (10AX115R3F40E2SG) で、浮動小数点の計算も可能で非常に高速に動作する。図 2.16 は LATOME ファームウェアの概要を表している。

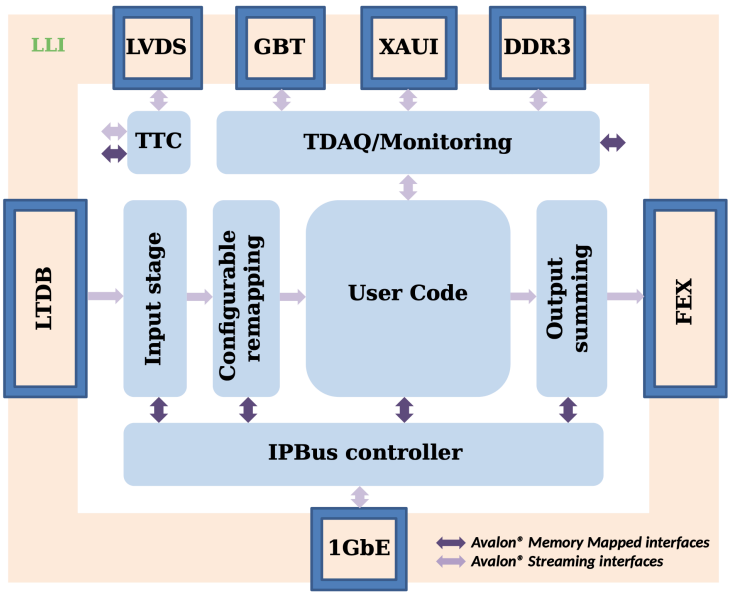


図 2.16: LATOME ファームウェアの概要 [19]。LLI は FPGA コアと外部信号を接続するインターフェイスとしての役割を果たす。

LTDB から受信される各 SC の ADC データは LATOME ファームウェアの LLI に入り、いくつかの block を経て FEX にエネルギー情報が送信される。表 2.5 に各 block のレイテンシをまとめた。

表 2.5: LATOME ファームウェアの各 Block の Latency (要求値) [19]

Block	Latency [ns]	BC
LLI: Deserializer of data from LTDB	50	2.0
Input Stage	75	3.0
Configurable Remapping	37.5	1.5
User Code	125	5.0
Output Summing	37.5	1.5
LLI: Serializer of data to FEX	50	2.0
LATOME Firmware (Total)	375	15.0

以下、各 block の役割を順に述べる。

Low-Level Interface (LLI)

LTDB からの各 SC の ADC データはまず初めに Low-Level Interface (LLI) に入る。LLI の目的は、FPGA コアに入出力されるデータのデシリアライズ/シリアライズや外部素子との通信である。これは一般に Board Support Package (BSP) として知られている。また、ファームウェア内の各 block の動作周波数 (320 MHz, 280 MHz, 240 MHz) を作り出して提供する役割も持つ。図 2.17 に LLI の出力データフォーマットを示す。

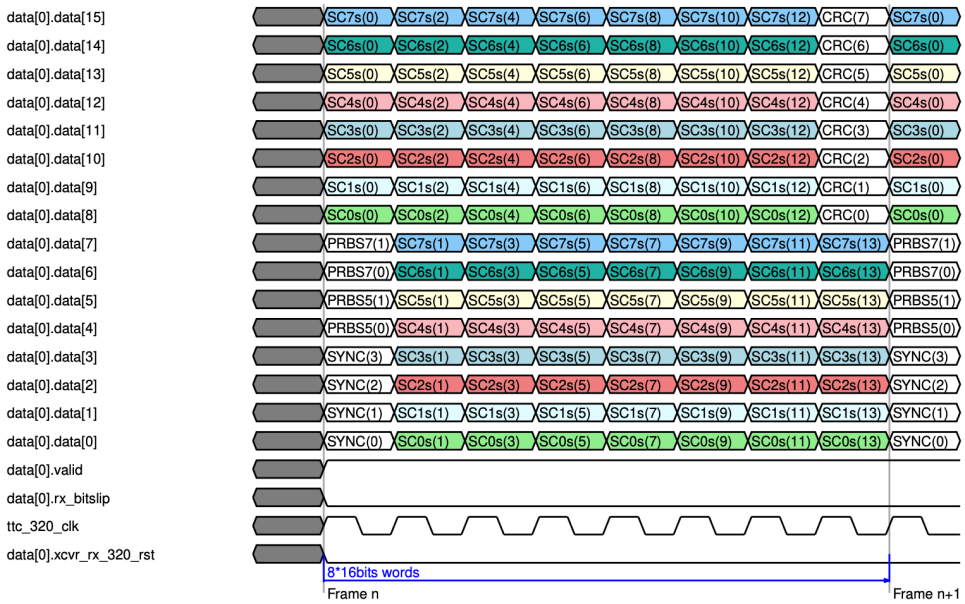


図 2.17: LLI の出力データフォーマット [19]。上は 1 本の光ファイバーからシリアルに入ってくる 8 個の SC の ADC データを 320 MHz のクロックに同期させてパラレルに変換した様子を示している。この 8×16=128 bit を 1 フレームと呼び、この 16 bit 幅のフレームの流れを 1 stream という。1 stream は最大で 8 SC (各 SC の 12 bit の ADC データは bit 単位で上図のように散りばめられる) であり、全部で 48 stream 存在する。

LTDB からの 1 本の光ファイバーには最大で 8 個の SC の ADC データが 40 MHz のクロックに同期してシリアルに送られてくる。LLI ではこれをパラレルにして、320 MHz のクロックに同期させて Input Stage に送る。このときの通信速度は、光ファイバー 1 本あたり、16 bit×320 MHz=5.12 Gbps になる。

Input Stage (IStage)

Input Stage (IStage) では、図 2.17 のデータ群からフレームの境界 (SYNC(0) から SYNC(3) の 4 bit) を検出し、それぞれの SC の ADC データを bit 番号順に並べる。また、BCID (PRBS5(0) から PRBS7(1) の 4 bit) の情報を抜き出し、対応する 12 bit の BCID を計算する。Input Stage は最終的に、12 bit の ADC データ、1 bit の Start of frame signal、12bit の BCID、1 bit の valid signal を 320 MHz のクロックで Configurable Remapping (Remap) に送る (図 2.18)。これも LLI と同様に 48 stream ある。

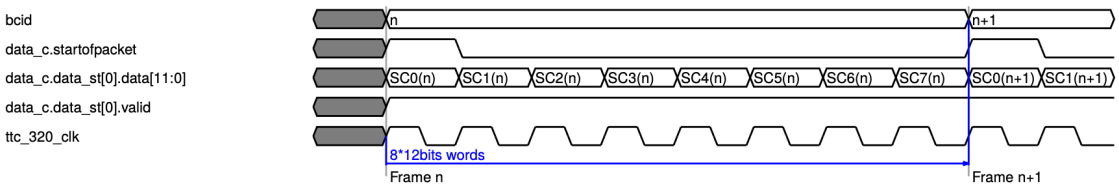


図 2.18: IStage の出力データフォーマット [19]。上は 1 stream の例であり、n は 12 bit の BCID を表す。1 stream は 8 個の SC を含み、1 クロックおきに対応する SC の 12 bit の ADC データが Remap に送られる。

Configurable Remapping (Remap)

Configurable Remapping (Remap) では、検出器の構造に依存 (すなわち、LATOME Board に依存) して、各 BC で IStage からの ADC データの並べ替えを行う。これは、後段の User Code の動作周波数が 240 MHz のため、1 stream あたり最大 8 個の SC を扱っていたところを最大 6 個にする必要があるからである[7]。並べ替えは同じ Trigger Tower に属する SC が 1 つの stream に集約されるように行う。これにより、後段で行う jFEX や gFEX のエネルギー計算が簡単になる。1 stream あたりの SC が減れば、stream の数は増える。Remap は 8 SC×48 stream を 6 SC×62 stream にして 240 MHz のクロックで User Code に ADC データを送信する (図 2.19)[8]。

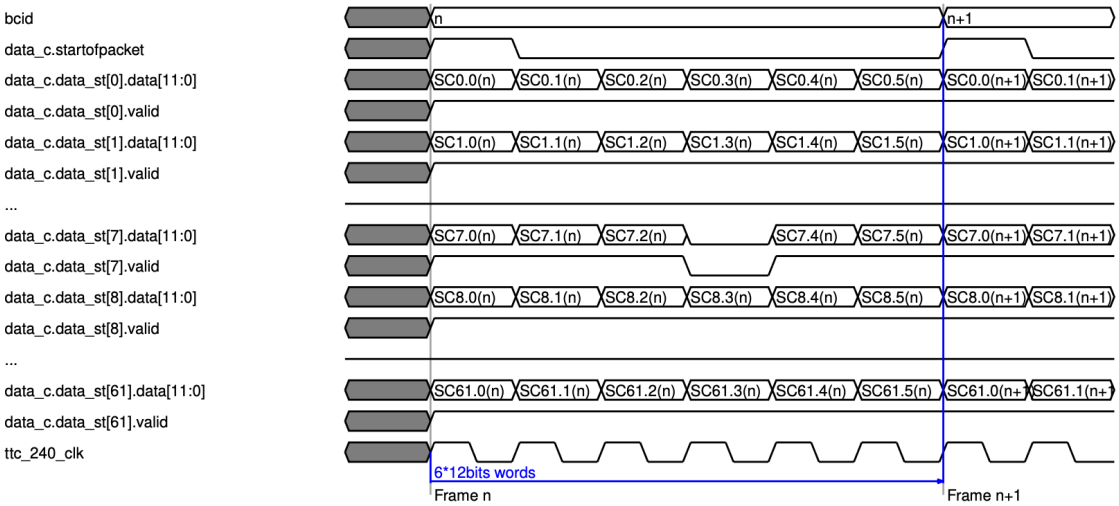


図 2.19: Remap の出力データフォーマット [19]。Stream 番号は 0 から 61 まである。例えば、SC0.0(n) は BCID n でサンプリングされたある SC の 12bit の ADC データが入っている。また、SC7.3(n) のような空データは valid が 0 である。SC0.X と SC1.X は同じ Trigger Tower に属している。

User Code

User Code は 112.5 ns のレイテンシで入射粒子のエネルギーとタイミングを再構成する LATOME ファームウェアで最も重要な block である。エネルギーとタイミングの再構成には Optimal Filter と呼ばれるフィルタリングアルゴリズムが使われている。また、再構成は 62 stream 並列で行い、240 MHz のクロックで動作するので 1 stream で 6 個の SC のエネルギー計算が可能である。図 2.20 に User Code の概要を示す。

[7] LHC のクロックは 40 MHz なので、1 stream に含まれる SC の数は、320 MHz であれば $320 \div 40 = 8$ SC、240 MHz であれば $240 \div 40 = 6$ SC と計算できる。

[8] 並べ替えの前後で総チャンネル数が異なる ($372 \neq 384$)。これは、1 つの LATOME で処理できる Super Cell の最大の数が 320 であることに起因している。つまり、少なくとも 50 から 60 のチャンネルは空データであり、失われた 12 チャンネルは必ず空データになるよう設計されている。

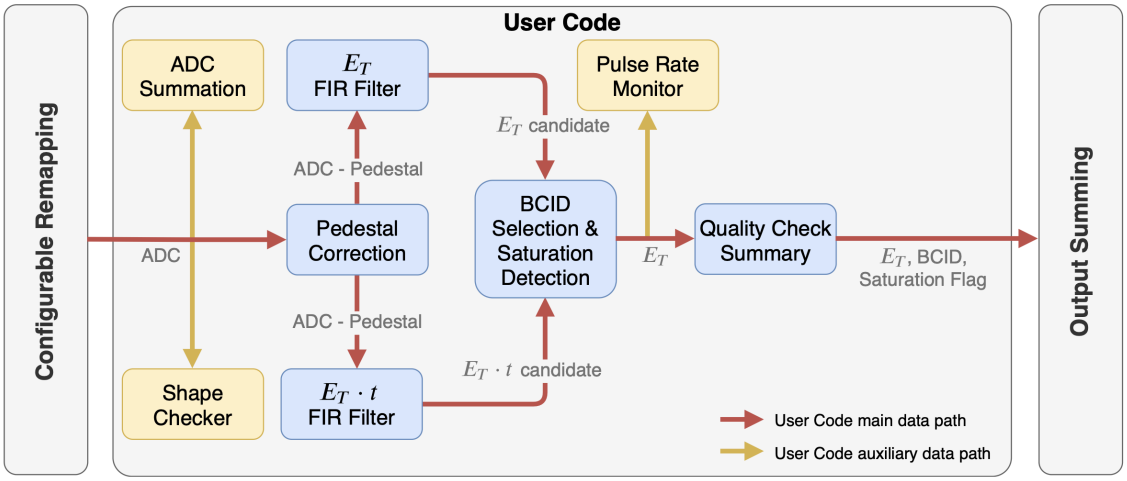


図 2.20: User Code block の概要 [18]。赤い矢印がメインパスで、Remap から受け取った 12 bit の ADC データからエネルギーとタイミングを再構成し、18 bit のエネルギーや BCID などの情報を 240 MHz のクロックで Osum に送る。

以下、メインパスにある各 sub block の役割を簡単に示す。

- **Pedestal Correction**：サンプリングされた 12 bit の ADC から Pedestal と Baseline を引く[9]。
- E_T&$E_T\tau$ **FIR Filter**：Optimal Filter により、対象の SC の横エネルギー E_T とタイミング情報 $E_T\tau$ の候補を計算 (τ は理想的なタイミングからのずれを表す)。
- **BCID Selection & Saturation Detection**：エネルギーとタイミングに関する制約を課して、正しいタイミング (BCID) でのみ横エネルギー E_T を採用し、それ以外はゼロとする。また、飽和した波形の同定も行う。
- **Quality Check Summary**：波形が飽和したかどうか、正しい横エネルギー E_T の BCID 同定と飽和した波形の BCID 同定ができたかを示す、4 bit の Quality flag を生成する。

User Code では各 SC の 4 点の ADC データに対して Optimal Filter を適用し、エネルギーとタイミングの再構成を行う。以下、そのアルゴリズムについて簡単に述べる。

[Optimal Filter]

液体アルゴンカロリメータに入射した粒子のエネルギーとバイポーラ波形の波高には線形性がある。したがって、25 ns ごとにサンプリングされた点をうまく使って、波高の高さを算出すればエネルギーが再構成できる。ただし、1 つのサンプリング点のみから求めるのはノイズなどの影響もあり精度が十分ではない。そのため、連続する 4 つのサンプリング点を用いて波高を求める。BC i の ADC の値を S_i、Baseline の値を B_i、Pedestal の値を p とすると再構成される横エネルギー E_T とタイミング情報 $E_T\tau$ は係数 a_i と b_i を用いて以下のように求められる。

[9] Pedestal は信号の基準点を指す。また、LHC ではトレイン構造により、連続するバンチの始まりと終わりではパイルアップによる波形が相殺されず、信号の基準点が上下にシフトしてしまう。これを Baseline shift という。したがって、User Code では 1024 個の ADC データの平均を求め約 10 秒おきに Baseline を更新、毎 BC 全ての SC において Baseline を減算してシフト分の補正を行う。

$$E_{\mathrm{T}} = \sum_{i=0}^{N-1} a_i(S_i - p - B_i) \tag{2.3}$$

$$E_{\mathrm{T}}\tau = \sum_{i=0}^{N-1} b_i(S_i - p - B_i) \tag{2.4}$$

サンプリング数 N は精度保証とレイテンシの兼ね合いで 4 に設定され、係数 a_i と b_i はノイズを最小にするような条件のもとで Lagrange の未定乗数法を解くことで求められる。これらの係数は実際に実験が始まる前に較正により決定される。このような方法を用いたフィルタリング機構が Optimal Filter である。したがって、係数 a_i と b_i はそれぞれ OFCa と OFCb と呼ばれる (Optimal Filter Coefficient)。User Code の FIR Filter sub block では FPGA 内の DSP と呼ばれる積和計算が可能な block を用いて毎 BC エネルギー計算が行われる。しかし、式 (2.3) と式 (2.4) で求められるのはあくまでエネルギーとタイミング情報の候補であって、本当に正しい値とは限らない。これに対処するため、以下に述べる tau criteria と呼ばれる選択機構を設けることで正しいエネルギーと BCID の同定を行う。

[tau criteria]

Optimal Filter は毎 BC エネルギー E_T とタイミング情報 $E_T\tau$ の候補を出力する[10]。しかし、粒子が検出器に入射したタイミング[11]はバイポーラ波形の立ち上がりの瞬間であり、この立ち上がりの瞬間とバイポーラ波形のピークを含む連続した 4 点で計算されたエネルギーこそが、本当に正しいエネルギーである。この正しいエネルギーと BCID の同定を行うための条件が tau criteria である。その内容は以下のようになっている。

$$\begin{cases} -8 < \tau < 14 \text{ ns}, & \text{for } E_{\mathrm{T}} > 10 \text{ GeV} \\ -8 < \tau < 8 \text{ ns}, & \text{for } E_{\mathrm{T}} \leq 10 \text{ GeV} \end{cases} \tag{2.5}$$

式 (2.5) を満たした (E_T, τ) のペアが正しい値であり、これらのペアが計算された BCID が正しいタイミング、そのときの E_T が正しいエネルギーである。反対に、式 (2.5) を満たさなかった BCID では E_T はゼロに設定され後段に送られる。

図 2.21 はある SC のある入射粒子に対するバイポーラ波形に Optimal Filter と tau criteria を実行した様子を示している。青点で示されているのが元の ADC データ点で 25 ns ごとにサンプリングされている。一方、緑点は式 (2.3) により求められた入射粒子の横エネルギー E_T の候補である。赤点は緑点の中で tau criteria を満たした点であり、正しいタイミングでの正しいエネルギーを示している。

[10] Optimal Filter の定義により、直接 τ を求めるのではなくまず $E_T\tau$ を求める。E_T が分かればビットシフトにより $E_T\tau$ から τ が求められる。

[11] 衝突点から生成粒子が光速で飛ぶと仮定して、対象の SC に到達するであろう時間をタイミング τ の基準点 (0 ns) としている。

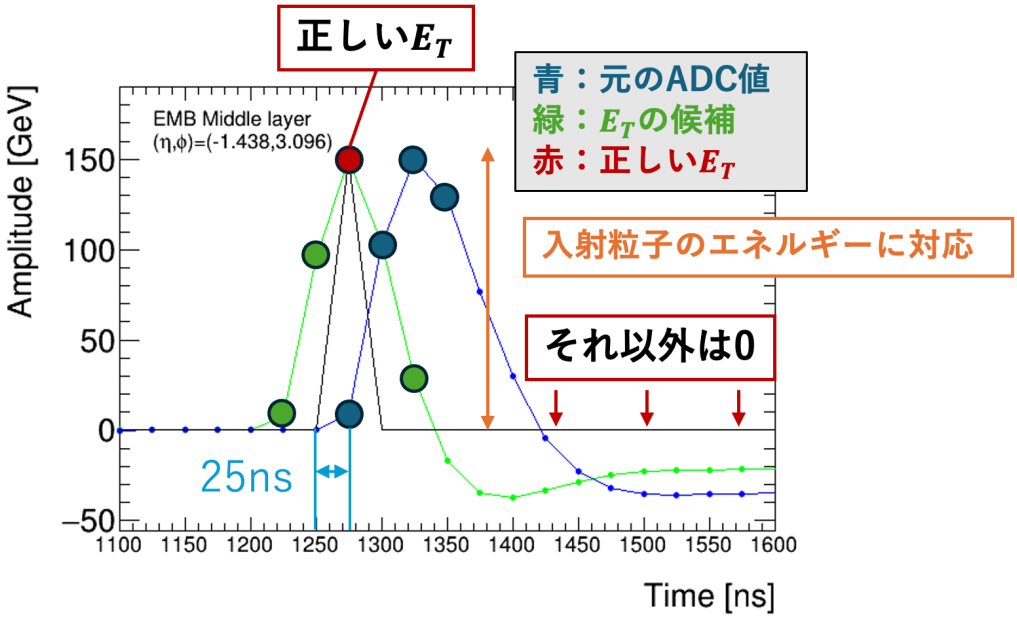


図 2.21: Optimal Filter によるエネルギー再構成。上は EMB の Middle layer のある SC のバイポーラ波形の例である。後段には、tau criteria を満たした E_T が正しい BC で送られ、満たさなかった BC では $E_T = 0$ として送られる。

最終的に、User Code は図 2.22 に示すようなデータフォーマットで後段の Output Summing (Osum) に、12 bit の BCID、1 bit の Start of packet、18 bit の E_T (LSB は 12.5 MeV に固定)、4 bit の Quality bit、1 bit の valid を 240 MHz のクロックに同期させて送信する。

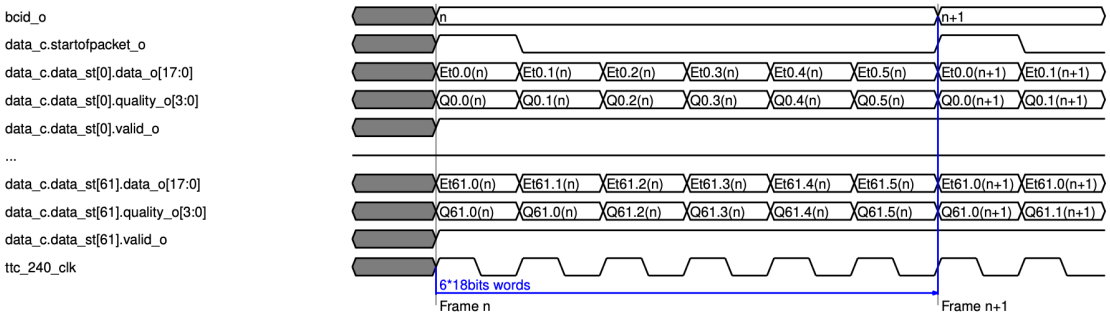


図 2.22: User Code の出力データフォーマット [19]。1 stream は 6 SC であり、62 stream 全てを並列に送信する。

Output Summing (Osum)

Output Summing (Osum) は LATOME ファームウェアで処理されたデータを L1Calo の FEX 向けに取りまとめる役割を持つ、本研究において最も重要な block である。具体的には、User Code で求められた SC ごとのエネルギーを表 2.4 で示した検出器のある領域 ($\Delta\eta \times \Delta\phi$) で足し合せ、トリガーのための 3 種類のクラスター情報 (eFEX, jFEX, gFEX) を LLI を経由して光ファイバーで FEX に送信する。図 2.23 に Barrel 領域における各 FEX の典型的なサイズを示す。jFEX は Trigger Tower 1 個分の大きさ、gFEX は Trigger Tower 4 個分の大きさである。

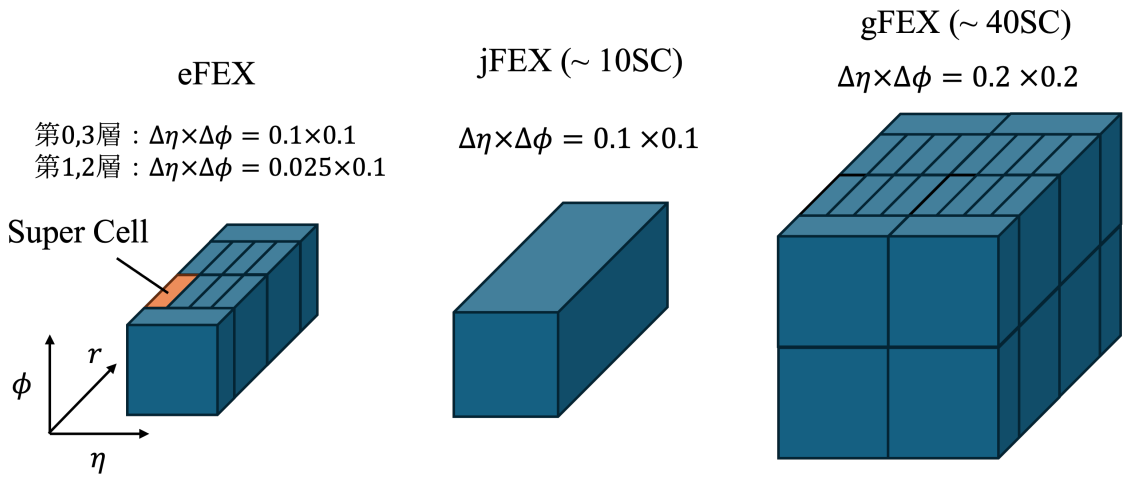


図 2.23: Barrel 領域の eFEX, jFEX, gFEX の典型的な大きさ。jFEX は典型的には 10 個の Super Cell、gFEX は典型的には 40 個の Super Cell を含む。

jFEX や gFEX のグルーピングは eFEX のデータを複製して使う必要がある。Osum ではこの複製処理も行う。このため、1 枚の LATOME Board に入力されるデータの通信速度 (約 200 Gbps) よりも出力されるデータの通信速度 (約 350 Gbps) の方が大きい。116 枚全ての LATOME Board を考慮すると、FEX に送信されるデータの通信速度は約 40 Tbps になる。

Osum は最終的に図 2.24 に示すようなデータフォーマットで 3 種類のクラスター情報を 280 MHz のクロックで FEX に送信する。

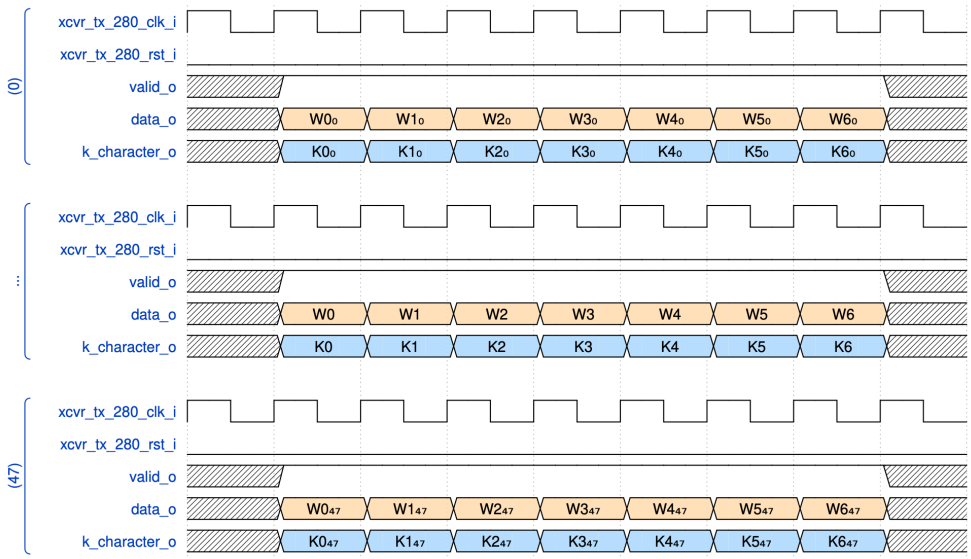


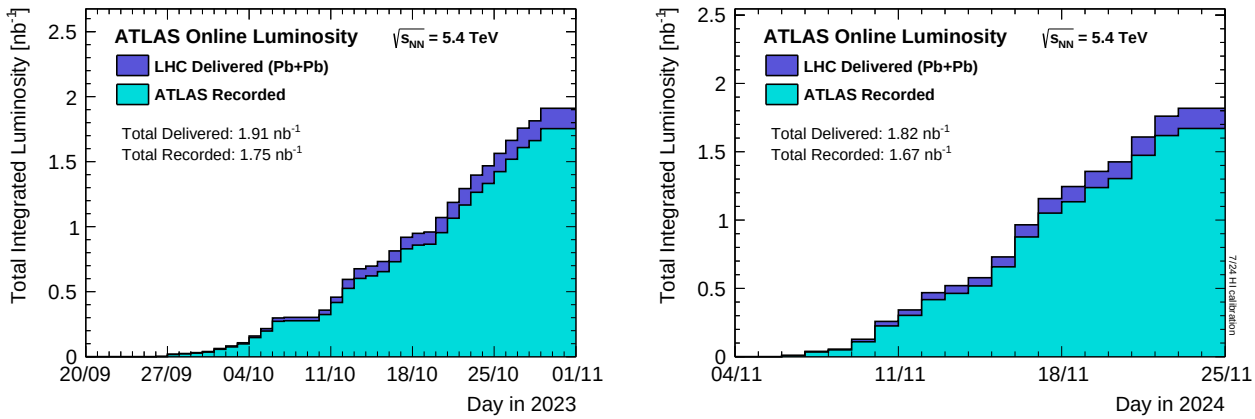
図 2.24: Osum の出力データフォーマット [19]。W は BCID や各 FEX のエネルギーなどを含んだ 32 bit 幅の FEX データ、K は 4 bit の信号同期等のための信号であり、全部で 48 stream ある。W には 4 bit の FEX ID も含まれており、0 が eFEX、1 が jFEX、2 と 3 が gFEX を表す。つまり、各 32 bit 幅の W は eFEX, jFEX, gFEX のいずれかに関するデータを持つ。

第3章　重イオン衝突実験向けのトリガー発行アルゴリズム

3.1　重イオン衝突実験

LHC では陽子・陽子衝突実験 (proton-proton run, pp run) に加えて、年に約 1 ヶ月間、重イオン衝突実験 (Heavy-ion run, HI run) も実施されている。重イオン衝突実験は、クォーク・グルーオンプラズマ (Quark-Gluon Plasma, QGP) と呼ばれる極限状態の物質の理解などを目的としている。QGP は宇宙誕生初期おおよそ 10^{-6} 秒から 10^{-5} 秒に存在したとされ、高温・高密度下でクォークとグルーオンが閉じ込めから解放され自由に運動できる状態である。LHC ではこの状態を一時的に生成し、ビッグバン直後の状態の再現と研究を可能にしている。重イオン衝突実験は主に ALICE 検出器が中心的な役割を果たしているが、ATLAS 検出器でもデータ取得が行われている。

衝突には鉛イオン (Pb) が用いられる[1]。鉛イオンを用いるのは、その大きな質量と核子数により、多数の陽子・中性子が衝突することで、超高温・超高密度の環境が形成されやすいからである。図 3.1 に、2023 年 9 月 26 日から 10 月 30 日と 2024 年 11 月 6 日から 11 月 25 日に実施された重イオン衝突実験において ATLAS 検出器で取得された積分ルミノシティを示す。その値はそれぞれ、1.75 nb^{-1} と 1.67 nb^{-1} で、重心系エネルギーはともに 5.4 TeV であった。



(a) 2023 年の HI run で取得された積分ルミノシティ　　**(b)** 2024 年の HI run で取得された積分ルミノシティ

図 3.1: Run 3 の重イオン衝突実験で取得された積分ルミノシティ [6]

鉛イオンは超相対論的な速度を持って衝突するので、ビーム軸方向に $1/\gamma$ のローレンツ収縮を起こす。そのため、衝突は必ずしも完全に重なり合うわけではなく、部分的な重なりや掠めるような衝突が発生する。これにより、中央衝突 (完全な重なり) では非常に高いエネルギー密度が生じる一方、超周辺衝

[1]鉛は 14 属の金属で、その原子番号は 82 である。また、原子量は 207.2 であり、全元素の中で最も重い安定同位体を持つ。Run で用いるのは質量数 208 (陽子 82、中性子 126、ともに魔法数) の鉛原子核で電子は完全に取り除かれる。

突 (Ultra Peripheral Collisions, UPC) では衝突の形状やエネルギー分布が異なる。次節では、この UPC とそれがもたらすデジタルトリガーの問題点について述べる。

3.2　デジタルトリガーにおける問題点

3.2.1　Ultra Peripheral Collisions (UPC) の物理

重イオン衝突に使われる鉛イオンは電荷が+82 e でかなり大きく、超相対論的な速度まで加速されるので非常に強力な電磁場を生成する。この電磁場は核子を構成するパートンや相手の核が持つ電磁場とも相互作用する。つまり、重イオン衝突ではハドロニックな相互作用に加えて電磁相互作用も起こる。これらの電磁相互作用は主に、超周辺衝突 (Ultra Peripheral Collisions, UPC) 中で起こる。具体的には、ビーム軸に垂直な平面内において 2 つの原子核間の距離が原子核半径の 2 倍を超える場合に起こり、これによりハドロニックな相互作用はほとんど抑制される [20]。

UPC の物理の例として、図 3.2 のような光子・光子散乱 ($\gamma\gamma \to \gamma\gamma$) が挙げられる。光子・光子散乱は、UPC で生じる非常に強い電磁場を介して 2 つの原子核が仮想光子を交換することで起こる。この事象を検出し精密に測定することは、量子電磁力学 (Quantum ElectroDynamics, QED) の検証に繋がる。また、光子・光子散乱の精密測定は Axion-Like Particles (ALPs) など標準模型を超える新粒子の探索にも繋がると期待されている。

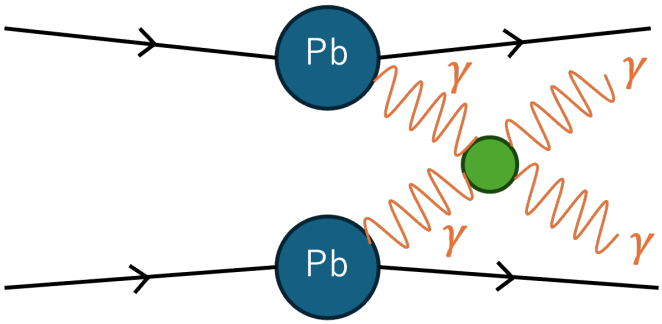


図 3.2: UPC 由来の光子・光子散乱。鉛イオンが持つ電磁場同士の相互作用により光子・光子散乱が誘発される。

UPC の光子・光子散乱による光子は、陽子・陽子衝突で生成される光子より比較的エネルギーが低いという特徴がある[2]。その理由は、鉛イオンの電磁場が生成する光子のエネルギースペクトルが低エネルギー側に偏っているためである[3]。液体アルゴンカロリメータではこの低いエネルギーの光子が検出されるが、そのエネルギーの低さゆえにトリガー発行に問題をもたらす。次項ではこれについて述べる。

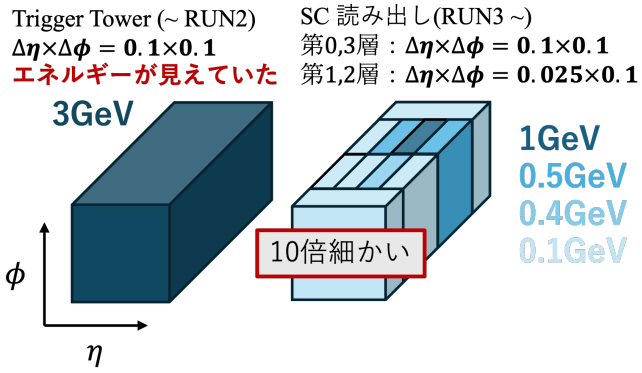
[2]UPC では $\gamma\gamma \to \gamma\gamma$ の他に $\gamma\gamma \to l^+l^-$ などの事象も起こる。液体アルゴンカロリメータでは主に終状態の電子が検出されるが、この電子のエネルギーも同様に低い。

[3]ローレンツ収縮により鉛イオンはディスク状に広がる。したがって、電磁場もディスク状に広がり、比較的長い波長 (低エネルギー) の光子が優勢となる (Weizsäcker-Williams 近似)。

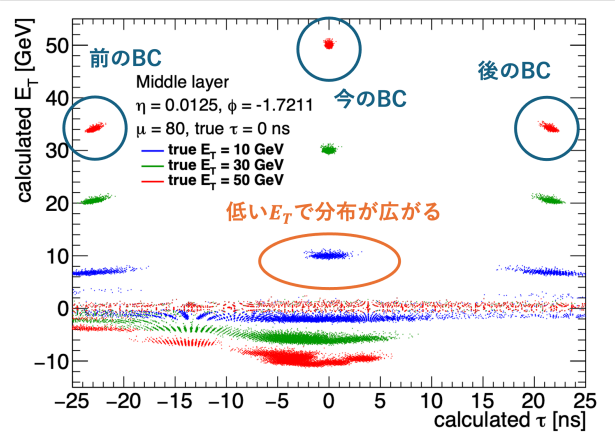
3.2.2　UPC がもたらすデジタルトリガーの問題点

上で述べたように、重イオン衝突実験における UPC では数 GeV 以下の低いエネルギーの事象が多い。その範囲は、液体アルゴンカロリメータのトリガーでおおよそ 2 GeV から 5 GeV である[4]。しかし、2023 年から本格的に運用を開始したデジタルトリガーシステムではこのような低いエネルギーの事象に対して、正しいタイミングでのトリガー効率が低いことが報告されている。

図 3.3a は、Run 2 までの Trigger Tower と Run 3 以降使われている Super Cell 読み出しについて、低エネルギー粒子が入射した場合のエネルギー読み出しの違いを表している。図は $E_T = 3$ GeV の粒子が入射した場合の例を示しており、Super Cell 読み出しは Trigger Tower と比較して 10 倍細かい構造を持つためエネルギーが分散され、セルあたりのエネルギーが小さくなっている。一方、図 3.3b は EMB の Middle layer にある SC の E_T と τ の相関を示している。これは陽子・陽子衝突データを用いているが、エネルギーが低くなればなるほどタイミング τ が広がりを持つようになるのが分かる。したがって、tau criteria の効率が下がり、SC のエネルギーは $E_T = 0$ に割り当てられてしまう。これは本来トリガーすべき信号がトリガーできていないことになる。実際に、現行の SC 読み出しでは 3 GeV 未満の低いエネルギーに対する BCID 同定効率が低く、図 3.3a に示したケースでは $E_T = 0$ となり信号が捨てられてしまう (図 3.4)。これに対処するため、現在、tau criteria に代わる全く新しい BCID 同定アルゴリズムの導入が求められている[5]。



(a) Trigger Tower と SC 読み出しの Low E_T の入射粒子に対する読み出し単位の違い



(b) EMB の Middle layer にある SC の E_T と τ の相関 [16]

図 3.3: UPC 由来の低エネルギー粒子がデジタルトリガーにもたらす影響

[4]陽子・陽子衝突の場合、興味のある事象が液体アルゴンカロリメータに落とすエネルギー E_T は典型的には 20 GeV 以上である。重イオン衝突の場合、その 4 分の 1 から 10 分の 1 程度しかない。

[5]FPGA のリソースの観点からあまり複雑な処理は実装できない。

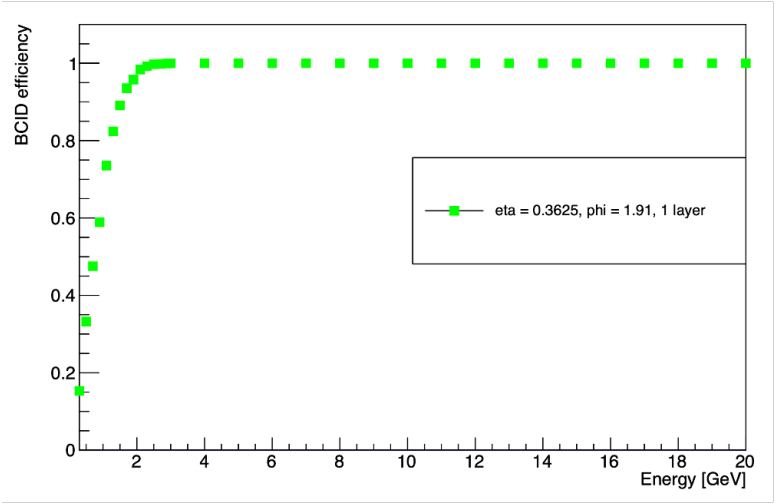


図 3.4: Barrel 領域のある SC における BCID 効率のエネルギー依存性 [21]

3.3 Peak Finding Algorithm による BCID の同定

Run 2 までの Trigger Tower による信号読み出しでは $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の領域内のアナログ信号を全て足した合わせた上で BCID を同定し、正しいエネルギーを算出することができていた。この経験から、重イオン衝突向けの BCID 同定アルゴリズムとして $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の領域から成る jFEX で Super Cell の信号を足し合わせ、ある程度エネルギーを大きくしてから BCID の同定を行う方法が提案された。その内容を図 3.5 に示す。

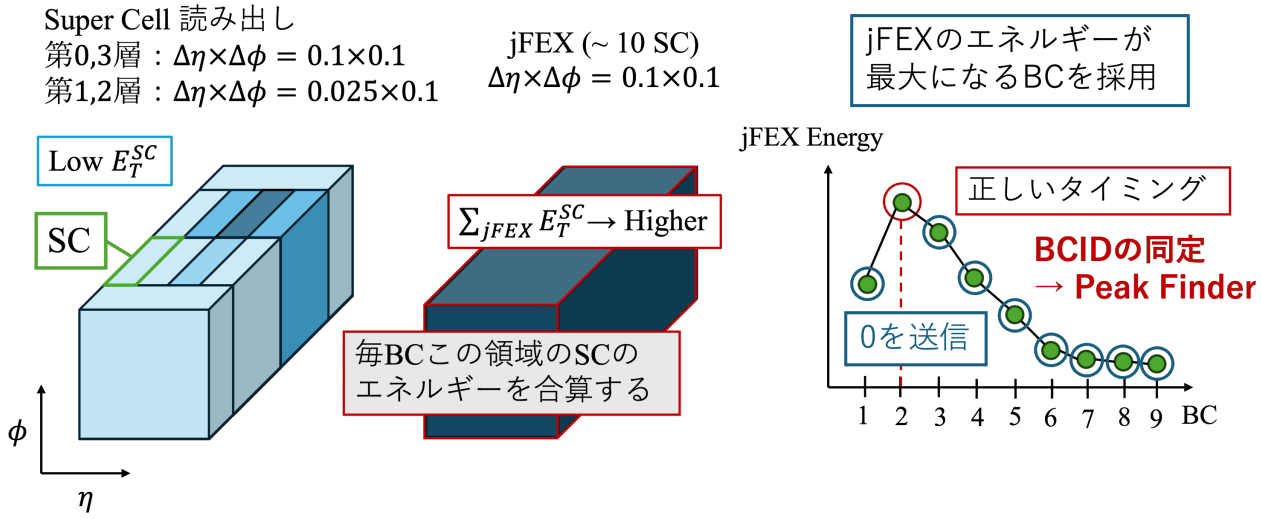


図 3.5: 重イオン衝突向けに考案された新しい BCID 同定アルゴリズム。E_T^{SC} は Optimal Filter によって再構成された個々の Super Cell のエネルギーを表す。Super Cell のエネルギーは jFEX の領域で毎 BC 足し上げられ、前後の BC と比較される。もし最大値をとればその BC が正しいとされ、そのときの Super Cell のエネルギーを後段に送る。

これは入射粒子が由来するタイミングとして jFEX のエネルギー ($E_T^{jFEX} = \Sigma_{\Delta\eta\times\Delta\phi=0.1\times0.1} E_T^{SC}$) が局所最大となる点を採用するアルゴリズムであり、Peak Finding Algorithm (PFA) と呼ばれている。以下に現在の BC を i とした場合の BCID 同定までの手順を示す。

1. BC i において、$\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の jFEX と呼ばれる領域で各 Super Cell のエネルギー E_T^{SC} を足し上げる。また、その値を $E_T^{jFEX}[i]$ とする。
2. $E_T^{jFEX}[i]$ の値を記憶する。
3. $E_T^{jFEX}[i-1]$ をその前後の BC の値、つまり $E_T^{jFEX}[i-2]$ と $E_T^{jFEX}[i]$ と比較する。
4. もし $E_T^{jFEX}[i-1]$ が最大 (ピーク) であれば、事象が起きたタイミングを BC $i-1$ と同定し、それに対応するエネルギー $E_T^{jFEX}[i-1]$ を後段に送信する。そうでなければゼロを送信する。
5. BC $i+1$ に進み、1 に戻って同じ処理を繰り返す。

上の手順を見て分かるように、今入ってきた jFEX のエネルギーがピークかどうか知るために、原理的に 1 BC のレイテンシが必要である。したがって、BC i におけるこのアルゴリズムの出力は BC $i-1$ の jFEX のエネルギーに対応するものになる。

この Peak Finding Algorithm は、「Peak Finder」という機能名で Osum に実装する予定である。Osum に実装する理由は、User Code で Optimal Filter によりエネルギーを計算した後、そのエネルギーを用いて信号を足し合わせ、BCID を同定する必要があるためである。なお、SC ごとの BCID の同定は行わないため、tau criteria は使用しない。本研究の目的は、この Peak Finder を High-Level Synthesis (HLS) を用いて LATOME ファームウェアの Osum に実装し、その動作の検証を行うことである。しかし、Peak Finder の開発に進む前に、ATLAS 検出器で実際に取得された重イオン衝突実験のデータを使ってこのアルゴリズムの有効性を検証することも重要である。次章では、その検証内容と結果について述べる。

第4章　実データを用いたPeak Finding Algorithmの検証

この章では、ATLAS 検出器で実際に取得された重イオン衝突実験データを用いた、Peak Finding Algorithm (PFA) の有効性の検証について述べる。前章の 3.3 節で述べたように、PFA では Super Cell のエネルギーを jFEX の領域 ($\Delta\eta \times \Delta\phi = 0.1 \times 0.1$) で足し上げる。この足し上げられたエネルギーに対して PFA を適用し、正しい BCID の同定ができるか、検証・評価を行うことがこの章の目的である。

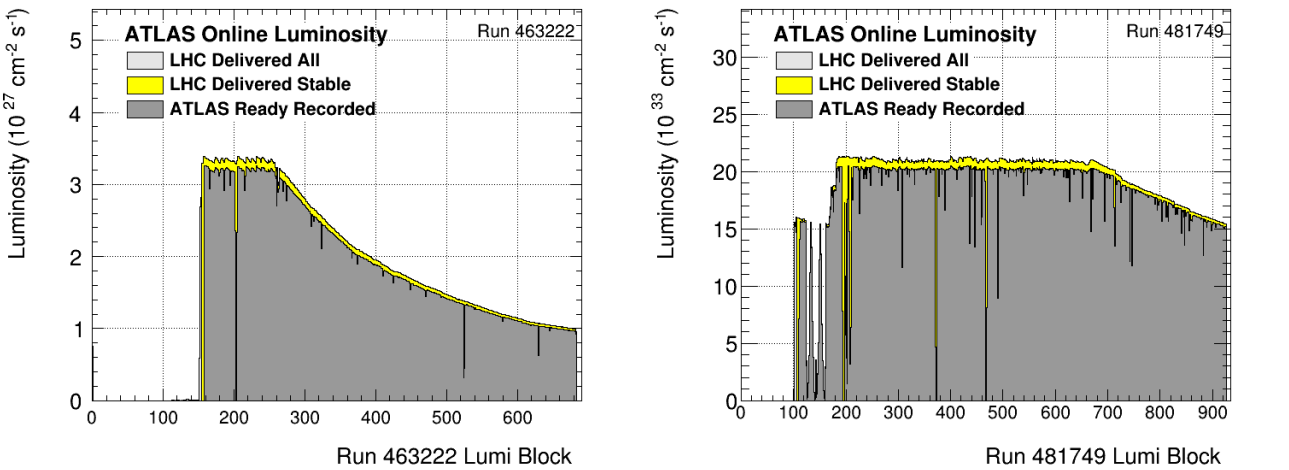
4.1　検証に使用したデータセット

PFA の検証には 2023 年秋に取得された重イオン衝突実験データを用いた。また、比較のため 2024 年夏に取得された陽子・陽子衝突実験データも用いた。どちらもデータのタイプは Express である。表 4.1 にその詳細を示す。

表 4.1: Peak Finding Algorithm の検証に用いた実データの詳細 [22]

Collision Type	Run Number	データ取得日	重心系エネルギー	Peak Luminosity	バンチ数
重イオン衝突	463222	2023.10.25	5.36 TeV	3.39×10^{27} cm^{-2} s^{-1}	880
陽子・陽子衝突	481749	2024.8.5〜8.6	13.6 TeV	2.13×10^{34} cm^{-2} s^{-1}	2,340

また、図 4.1 に各 run のルミノシティの時間変化を示す。Lumi Block (LB) は時間の単位で、1 LB は約 1 分である。



(a) 重イオン衝突データ (run 463222) のルミノシティの時間変化　**(b)** 陽子・陽子衝突データ (run 481749) のルミノシティの時間変化

図 4.1: Peak Finding Algorithm の検証に用いた HI と pp 衝突データのルミノシティの時間変化 [22]

4.2 イベントの選択条件と検証の方法

4.2.1 データ形式

上に挙げた実データはそのままの状態では使えないので、検証のためのデータ解析では形式を整える必要がある。本研究では、まず LAr Prompt Analysis というソフトウェアツールを用いて、実データに含まれる BCID やエネルギーなどの情報を branch としてまとめ、TTree を作り、root ファイルに保存した。この過程で、重イオン衝突と陽子・陽子衝突のデータはそれぞれ 127 個と 30 個のファイルにまとめられ、それぞれ 1 ファイルあたり数百から数千のイベントが含まれている[1]。さらに、1 イベントには 34,048 個の Super Cell の情報が含まれている。これらの情報は branch として保存され、名前が付けられる。表 4.2 に解析に用いた主な branch とその内容を示す。データ型は 1 つの Super Cell ではなく 1 つのイベントを基準にしている。したがって、例えばある Super Cell は 1 つの IEvent や lb、3 つの of_et と 3 つの of_ettau、1 つの sc_channelId や sc_layer などを持つ。なかでも、of_et は解析において最も重要な branch であるが、各 Super Cell につき 3 BC 分しかエネルギーが入っていないことに注意する必要がある。この branch の説明は 4.2.3 項で述べる。

表 4.2: 解析に用いた主な branch とその内容

Branch Name	Data Type	内容
IEvent	ULong64_t	イベントを識別するための番号
LArEventBits	UInt_t	LAr エレクトロニクスの異常を知らせるフラグ
lb	Short_t	Lumi Block
*of_et	vector<vector<int>>	Optimal Filter で再構成された SC の E_T^{SC} の候補 (3 BC/SC)
*of_ettau	vector<vector<int>>	Optimal Filter で再構成された SC の $E_T^{SC}\tau$ の候補 (3 BC/SC)
*sc_badword_UPD	vector<unsigned int>	SC の異常を知らせるフラグ
*sc_badword_UPD_cells	vector<unsigned int>	LAr cell の異常を知らせるフラグ
*sc_channelId	vector<int>	SC を識別するための ID
*sc_det	vector<short>	SC が所属するサブ検出器の ID (0=EMB, 1=EMEC, 2=HEC, 3=FCal)
*sc_eta	vector<float>	SC の η 座標
*sc_phi	vector<float>	SC の ϕ 座標
*sc_layer	vector<short>	SC が所属する layer の ID (0=Presampler, 1=Front, 2=Middle, 3=Back)
*sc_sum_main	vector<float>	メイン読み出しで計算された SC のエネルギー (横エネルギー換算前)
trig_calo	Bool_t	カロリメータトリガーのフラグ
trig_muon	Bool_t	ミューオントリガーのフラグ

root ファイルに保存された実データの情報は ROOT[2]で解析し、グラフの生成、ファイルの保存など

[1] それぞれの衝突データに含まれる Super Cell の総数 (統計量) が近い値になるようにこのようなファイル数にした。

[2] ROOT は CERN によって開発されたデータ解析フレームワークで、高エネルギー物理学を中心に広く利用されている。高速なデータ処理、ヒストグラム作成、関数フィット、統計解析、グラフによる可視化が可能で、C++をベースにしたスクリプト言語で記述される。

を行った。

4.2.2 イベント選択の条件

root ファイルに保存した実データはカットがかかっているとはいえ、非常に膨大なデータを含んでいる[3]。なかには、調子の悪い SC やノイズの大きなイベントも存在する。PFA の検証ではこれらの SC やイベントは解析から外した。また、目的に応じて解析に必要のない情報を切り捨てたり、トリガー条件の設定を行った。

以下に解析に用いたデータに課した条件を述べる。

- **LArEventBits = 0**：液体アルゴンカロリメータのエレクトロニクスに異常がない場合、この branch のフラグは 0 になる。したがって、この条件を課した。
- **Good Run List**：Good Run List (GRL) は物理解析に使える質の良いデータのリストである[4]。解析ではその範囲の Lumi Block を使用した。具体的には、run 463222 の重イオン衝突で 163 から 242、そして 245 から 683 である。一方、run 481749 の陽子・陽子衝突はデータが新しいためその範囲が公開されていなかった。よって、念の為、解析では 110 から 926 まで全ての範囲を使用した。
- **トリガー条件**：カロリメータトリガーによるバイアスを可能な限り排除するため、カロリメータ全体でトリガーがかかっていない事象を選択した。この条件は trig_calo=0 で指定できる。また、trig_muon=1 でミューオントリガーを要求し、カロリメータ全体に数 GeV 以上のエネルギーが存在する事象を選別した。これにより、液体アルゴンカロリメータに落とされた低いエネルギーの挙動が検証できる。
- **異常なイベントの排除**：run 463222 の重イオン衝突データにはノイズともとれる異常な振る舞いを示すイベントが少なからず存在していた。よって、これらのイベントの ID リストを作り、解析から外した。

これらの条件を通過したイベントは重イオン衝突データで 18,845 イベント、陽子・陽子衝突データで 17,146 イベント存在した (表 4.3)。さらに、これらのイベントから下に示す Bad SC や Bad jFEX を除いて解析を行った。

- **Bad SC**：波形が歪んでいたり信号の応答が無いなど状態の悪い Super Cell (Bad SC) は解析から外した。Bad SC は sc_badword_UPD と sc_badword_UPD_cells が共に 0 であるという条件を適用することで特定できる。そのような Super Cell は重イオン衝突データに 1,500/32,128 存在した[5]。一方、陽子・陽子衝突データには 1,193/32,128 存在した。

[3]本研究の解析で用いた重イオン衝突実験データの root ファイル (計 127 個) のサイズは、約 1.1 TB である。
[4]GRL には Beam が安定して各検出器が正常に動作している Lumi Block の範囲が公開されている。
[5]解析では 34,048 個全ての Super Cell を使用できなかった。これは構築できる jFEX のリストが不足していたからである。実際のデータ取得では全ての Super Cell がいずれかの jFEX に属する。

- **Bad jFEX**：Bad SC と同様に、状態の悪い jFEX (Bad jFEX) も解析から外した。Bad jFEX の定義は、その中に少なくとも一つの Bad SC を含む jFEX である。そのような jFEX は重イオン衝突データに 1,016/5,376 存在した[6]。一方、陽子・陽子衝突データには 808/5,376 存在した。図 4.2 に重イオン衝突データに含まれていた Bad jFEX の $\eta - \phi$ 分布を示す。

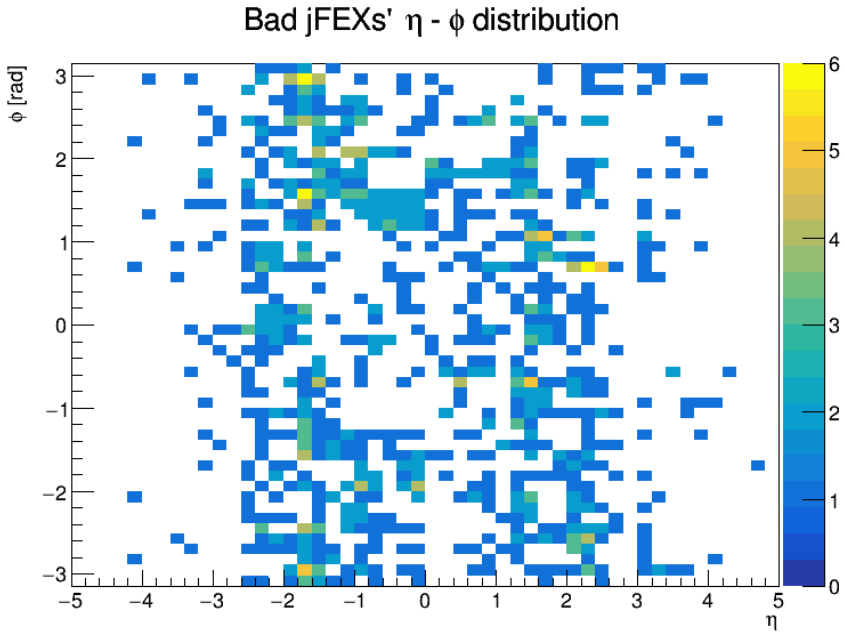


図 4.2: Bad jFEX の $\eta - \phi$ 分布 (HI run)

表 4.3: 解析で使われたファイル数とイベント数

	重イオン	陽子・陽子
ファイル数	127	30
イベント数	18,845	17,146
SC の数 (Bad SC を除く)	463,360,860	455,346,322
jFEX の数 (Bad jFEX を除く)	82,164,200	78,322,928

また、解析に使った実データの持つ基本的特徴をまとめておく。表 4.4 に解析で使われた Super Cell のエネルギー別の数、表 4.5 に $main\ E_T^{SC}$ があるエネルギー以上の Super Cell を少なくとも一つ含む jFEX の数を示す[7]。

[6]解析では 34,048 個全ての Super Cell を使用できなかったため、構築される jFEX も第 2 章の表 2.4 で示した 5,760 にわずかに届かない。

[7]$main\ E_T^{SC}$ は、Super Cell のメイン読み出し横エネルギーを意味し、sc_sum_main と sc_eta という branch を用いて、sc_sum_main / cosh(sc_eta) で求められる。ただし、単位は MeV である。

表 4.4: 解析で使われた Super Cell のエネルギー別の数

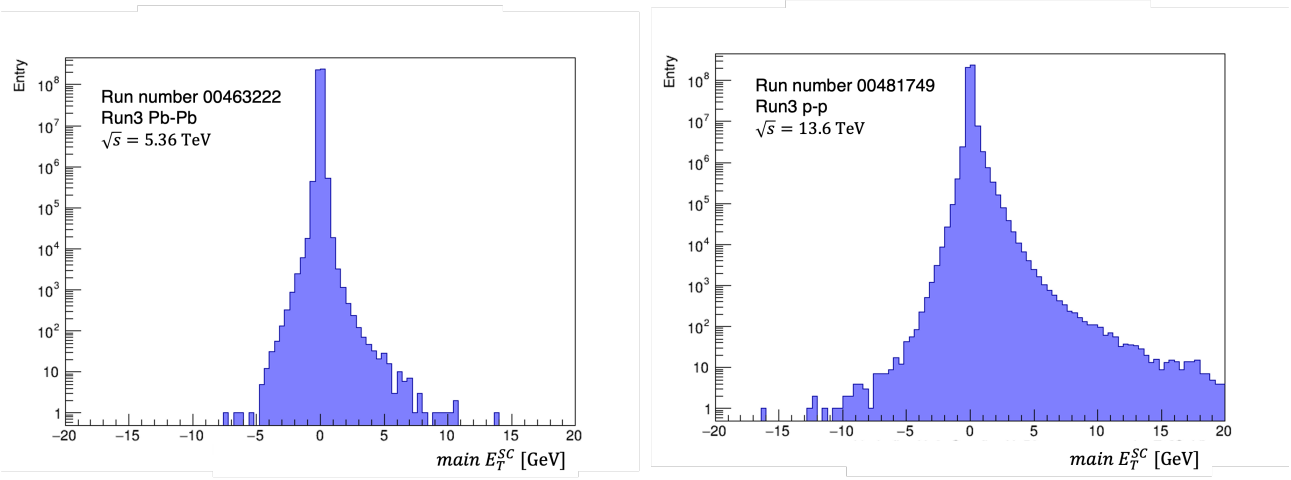
$main\ E_T^{SC}$	重イオン	陽子・陽子
0 GeV 以上	236,939,628	245,834,489
0.2 GeV 以上	3,796,545	33,195,520
0.4 GeV 以上	546,178	11,101,898
0.6 GeV 以上	95,016	5,438,361
0.8 GeV 以上	24,290	3,223,607
1 GeV 以上	10,112	2,087,373
5 GeV 以上	70	7,587
10 GeV 以上	4	692

表 4.5: $main\ E_T^{SC}$ がある値以上の SC を持つ jFEX の数

$main\ E_T^{SC}$	重イオン	陽子・陽子
0 GeV 以上	66,159,046	64,521,017
0.2 GeV 以上	3,770,746	26,251,557
0.4 GeV 以上	545,001	10,468,574
0.6 GeV 以上	94,845	5,304,199
0.8 GeV 以上	24,239	3,175,642
1 GeV 以上	10,096	2,064,612
5 GeV 以上	70	7,545
10 GeV 以上	4	690

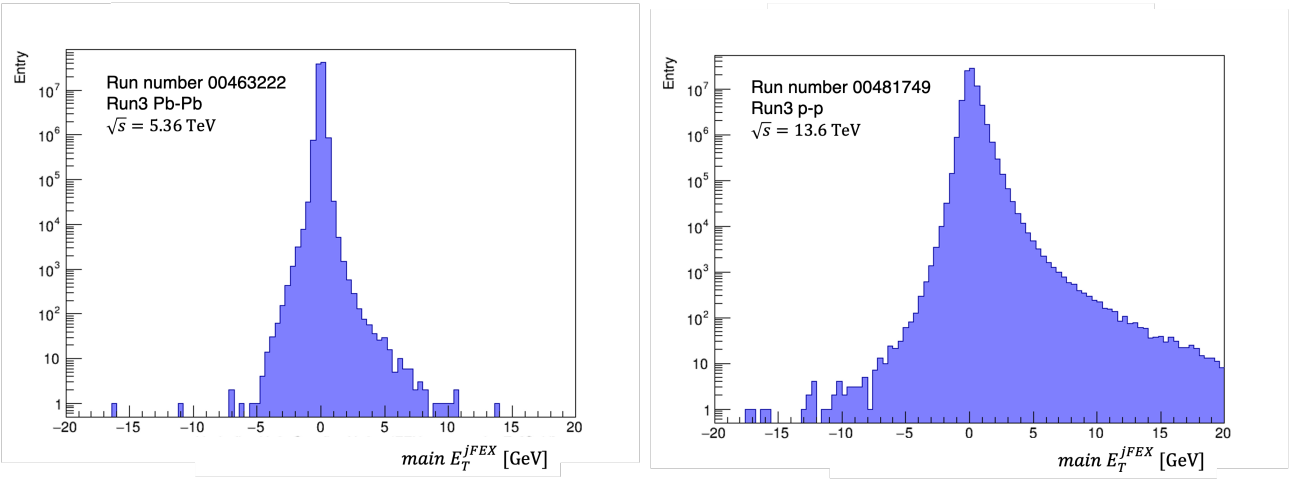
図 4.3 に解析で用いた Super Cell のメイン読み出し横エネルギー $main\ E_T^{SC}$ の分布、図 4.4 に解析で用いた jFEX のメイン読み出し横エネルギー $main\ E_T^{jFEX}$ の分布を示す[8]。

[8] $main\ E_T^{jFEX}$ は、その jFEX に含まれる各 SC の $main\ E_T^{SC}$ の和である。



(a) 重イオン衝突　　**(b)** 陽子・陽子衝突

図 4.3: 解析で用いた Super Cell のメイン読み出し横エネルギー分布



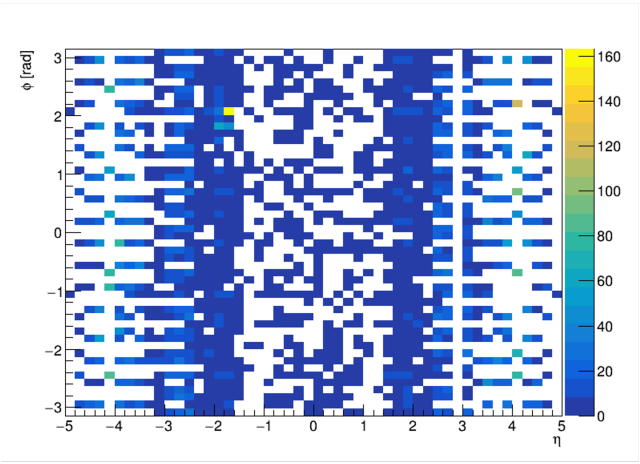
(a) 重イオン衝突　　**(b)** 陽子・陽子衝突

図 4.4: 解析で用いた jFEX のメイン読み出し横エネルギー分布

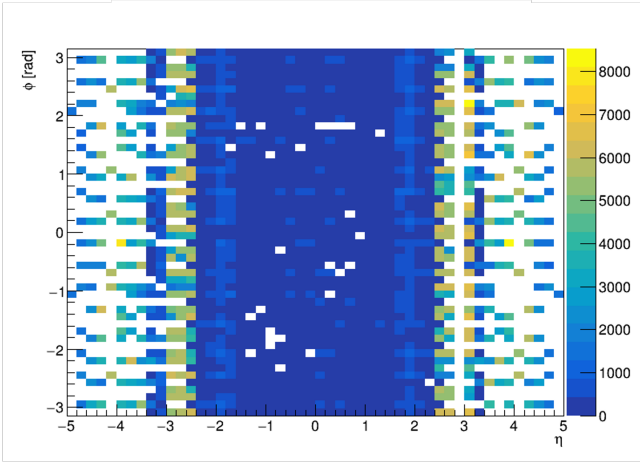
図 4.3 を見ると、重イオン衝突で液体アルゴンカロリメータに落とされるエネルギーは、陽子・陽子衝突の場合と比較して低いことが確認できる。その値は大きくても 10 GeV 程度である。また、図 4.4 が示すように jFEX の領域で Super Cell のエネルギーを足し上げた場合、分布の形状に大きな変化は見られなかった。

図 4.5 に $main\ E_T^{SC}$ が 1, 5, 10 GeV 以上の SC の $\eta - \phi$ 分布を示す。Bad SC は含まれていない。重イオン衝突では 1 GeV から 5 GeV のエネルギーを持つ Super Cell が多く分布し、$1.5 < |\eta| < 2.5$ の比較的前方領域が特に目立つ。一方、陽子・陽子衝突は 10 GeV 以上の Super Cell も所々確認されている。それらにおいても、やはり $2.0 < |\eta| < 3.0$ の前方領域が特に目立つ[9]。

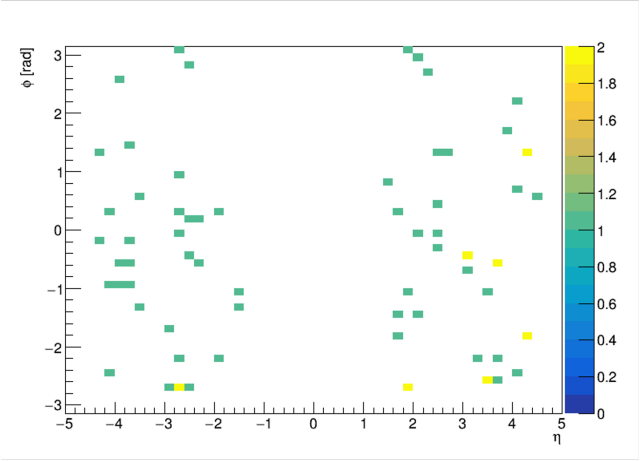
[9]HEC や FCal の SC の $\Delta\phi$ は 0.2 や 0.4 のものも存在するため、ビン幅の関係で $|\eta|$ の大きな領域には穴のように見える部分がある。



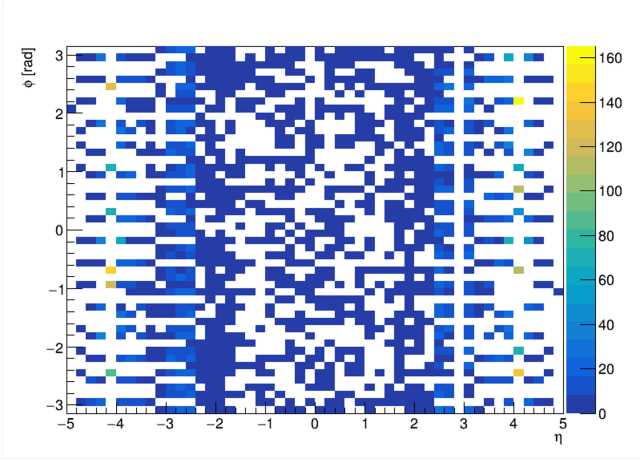
(a) *main* $E_T^{SC} \geq 1$ GeV の SC (HI run)



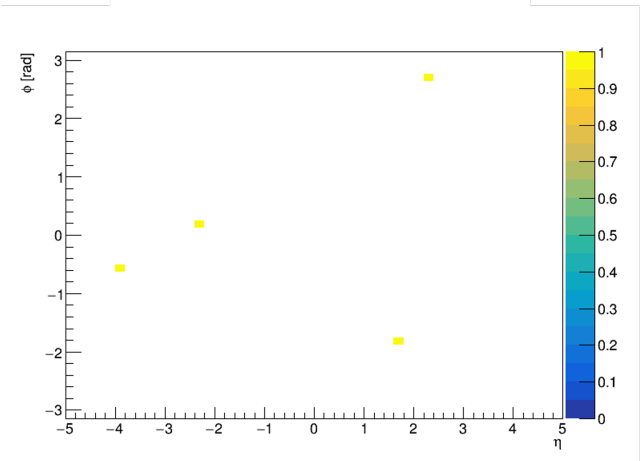
(b) *main* $E_T^{SC} \geq 1$ GeV の SC (pp run)



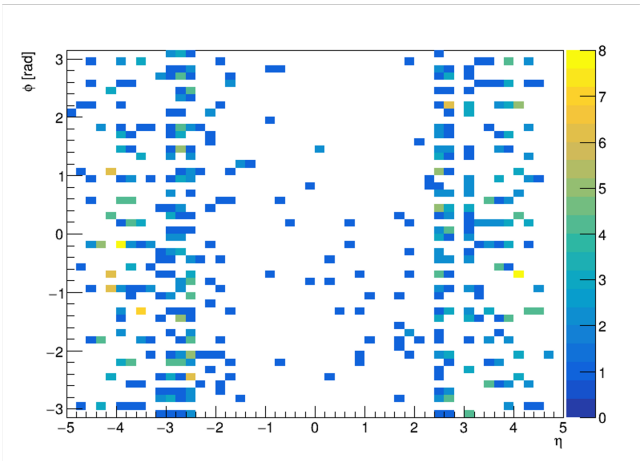
(c) *main* $E_T^{SC} \geq 5$ GeV の SC (HI run)



(d) *main* $E_T^{SC} \geq 5$ GeV の SC (pp run)



(e) *main* $E_T^{SC} \geq 10$ GeV の SC (HI run)



(f) *main* $E_T^{SC} \geq 10$ GeV の SC (pp run)

図 4.5: 有意なエネルギーを持つ Super Cell の $\eta - \phi$ 分布

4.2.3　Peak Finding Algorithm の検証方法

PFA の検証には of_et という branch を使う。この branch は Optimal Filter によって計算された Super Cell のエネルギー E_T^{SC} の候補が入った要素数 3 の vector である。実際のデジタルトリガーでは毎 BC エネルギー計算を行うが、この branch にはトリガーされた BCID を中心に、前後 1 BC 分のエネルギー

のみが格納されている。また、重イオン衝突の場合は平均相互作用数 μ は 1 よりはるかに小さいので、隣の BC で衝突が起こる可能性はほとんどゼロである[10]。したがって、図 4.6 のように、Super Cell 単体を考えれば、該当イベントに電子や光子の信号が含まれている場合、index = 1 でエネルギーがピークになることが期待される。以降、of_et を E_T^{SC} と呼び、index を i $(i = 0, 1, 2)$ で指定する。

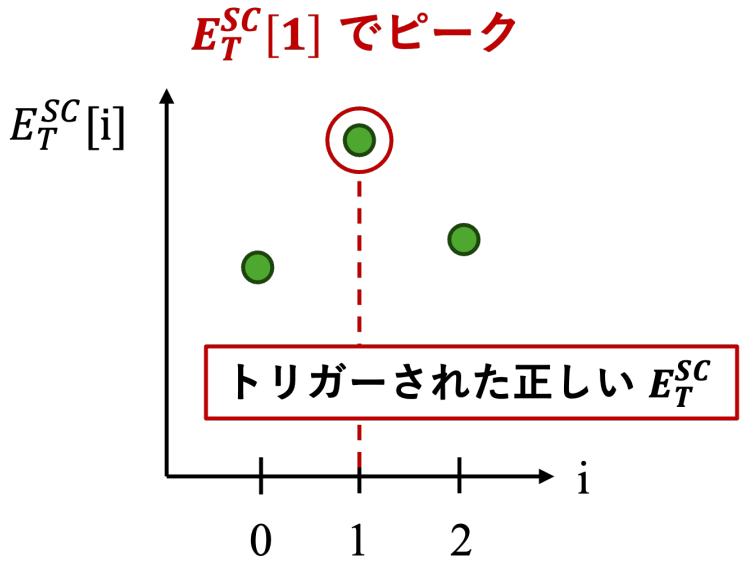


図 4.6: 期待される E_T^{SC} の index 分布

以下に検証のステップを示す。ただし、上で述べたイベント選択条件は適用済みとする。

1. あるイベントに含まれる全ての Super Cell の情報を取得する。
2. そのうち、32,128 個の Super Cell を用いて 5,376 個の jFEX を構築する。
3. 各 jFEX に含まれる各 Super Cell が持つ $E_T^{SC}[i]$[11]を BC ごと (つまり、index ごと) に足し上げ、jFEX のエネルギー $E_T^{jFEX}[i]$ を計算する。同様に、各 Super Cell が持つメイン読み出し横エネルギー $main\ E_T^{SC}$ を jFEX の領域内で足し上げ、jFEX のメイン読み出し横エネルギー $main\ E_T^{jFEX}$ も計算する。
4. jFEX に含まれる Super Cell の $main\ E_T^{SC}$ や jFEX 自身の $main\ E_T^{jFEX}$ の条件を適宜変えながら、$E_T^{jFEX}[0]$, $E_T^{jFEX}[1]$, $E_T^{jFEX}[2]$ の大小を比較し、最大になる index の分布を作る。つまり、横軸 index、縦軸エントリー数のヒストグラムになる。また、横軸が $E_T^{jFEX}[i]$ を最大にする index、縦軸がその jFEX のメイン読み出し横エネルギー $main\ E_T^{jFEX}$ の 2 次元ヒストグラムも作る。
5. 1 に戻り、全ファイル・全イベントで同様の処理を繰り返す。
6. UPC の物理で懸念される 2 GeV から 5 GeV の低いエネルギー領域で jFEX のエネルギー $E_T^{jFEX}[i]$ が index = 1 でピークを持つものがどのくらい存在するか確かめる (BCID 同定率の計算)。

基本的なステップは上に示した通りだが、これ以外にも検証に必要なグラフをいくつか作成した。

[10]本研究で用いた重イオン衝突データ (run 463222) では、$\mu = 3 \times 10^{-5}$ であった。
[11]i 番目の BC と仮定した時の SC の再構成エネルギーを意味する。

4.3 Peak Finding Algorithm の検証

4.3.1 E_T^{jFEX} を最大にする index 分布

重イオン衝突データの解析で構築された全ての jFEX について、E_T^{jFEX} を最大にする index の分布を図 4.7 に示す。この分布は Super Cell や jFEX のエネルギーに何の条件も課していない。ここで、各 index の定義を表 4.6 に示す。図 4.7 の index 分布では、$E_T^{jFEX}[1]$ でピークをとる jFEX が最多ではない。しかし、これは負のエネルギー領域も含んでおり、非常にエネルギーが低い jFEX など実際のトリガーにはかからないものも多く存在することに注意したい。

表 4.6: Index 分布中の各 index の定義

Index	定義
0	$E_T^{jFEX}[0]$ が最大の jFEX
1	$E_T^{jFEX}[1]$ が最大の jFEX
2	$E_T^{jFEX}[2]$ が最大の jFEX
3	$E_T^{jFEX}[0] = E_T^{jFEX}[1]$ かつ $E_T^{jFEX}[2]$ がそれ未満の jFEX
4	$E_T^{jFEX}[0] = E_T^{jFEX}[2]$ かつ $E_T^{jFEX}[1]$ がそれ未満の jFEX
5	$E_T^{jFEX}[1] = E_T^{jFEX}[2]$ かつ $E_T^{jFEX}[0]$ がそれ未満の jFEX
6	$E_T^{jFEX}[0] = E_T^{jFEX}[1] = E_T^{jFEX}[2]$ の jFEX

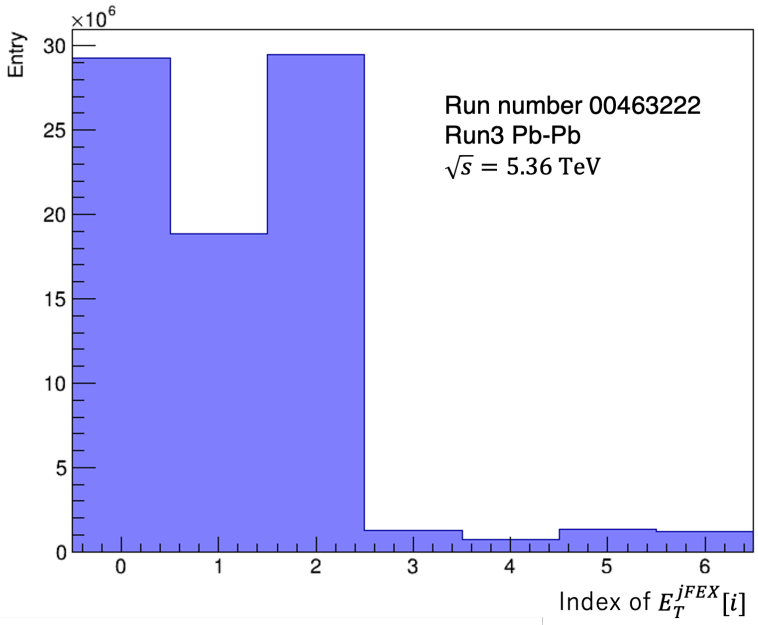
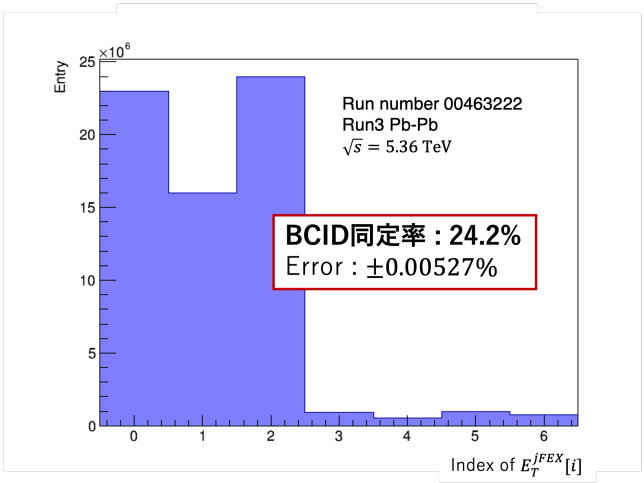


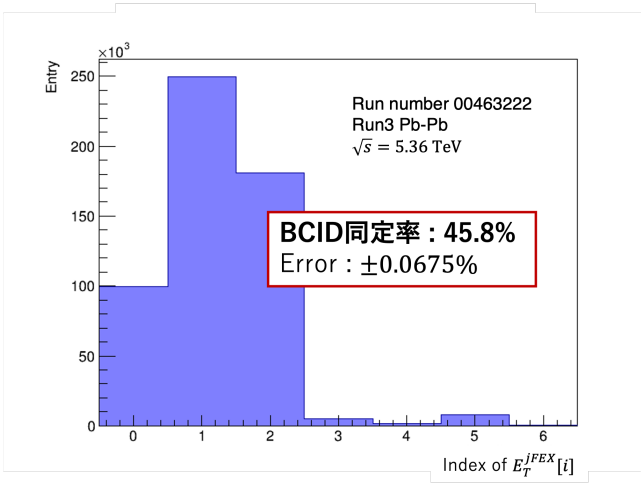
図 4.7: E_T^{jFEX} を最大にする index 分布 (HI run)

次に、jFEX を構成する Super Cell のメイン読み出し横エネルギー $main\ E_T^{SC}$ をベースに、図 4.7 にカットをかけた index 分布も作成した。具体的には、$main\ E_T^{SC}$ が 0, 0.4, 0.8, 1, 5, 10 GeV 以上の Super Cell を少なくとも一つ含むような jFEX だけを抽出して、同様の index 分布を作った。図 4.8 に重イオン

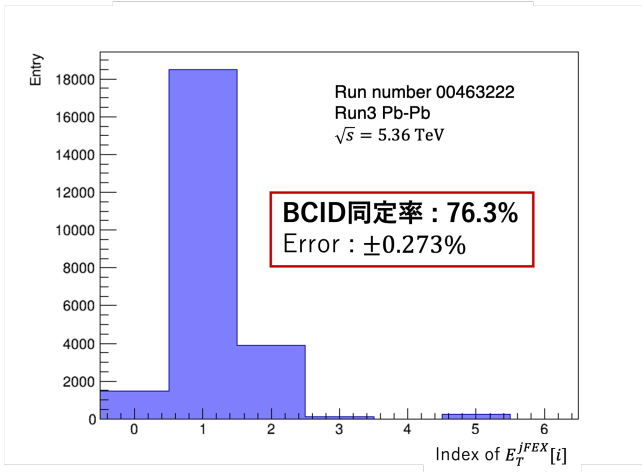
衝突データの場合の分布を示す。Index の定義は上で説明したものと同じである。また、全エントリー数のうち E_T^{jFEX}[1] が最大になった jFEX の割合 (これを BCID 同定率と定義する) もヒストグラム中に示した。$main\ E_T^{SC}$ が 0.4 GeV 以上で、index = 1 でピークをとる jFEX が最多になり始める。そのときの BCID 同定率は 45.8% (統計誤差は ±0.0675%) であった。この傾向は、$main\ E_T^{SC}$ が大きくなるにつれて顕著になり、$main\ E_T^{SC}$ が 1 GeV 以上の SC を含む jFEX においては、BCID 同定率は 85.9% (統計誤差は ±0.346%) に達した。このことは、Peak Finding Algorithm が有効である可能性を示唆している。一方、$main\ E_T^{SC}$ が 10 GeV 以上になると BCID 同定率は 50% (統計誤差は ±25.0%) になった。しかしこれは単に統計量が不足しているからだと考える。



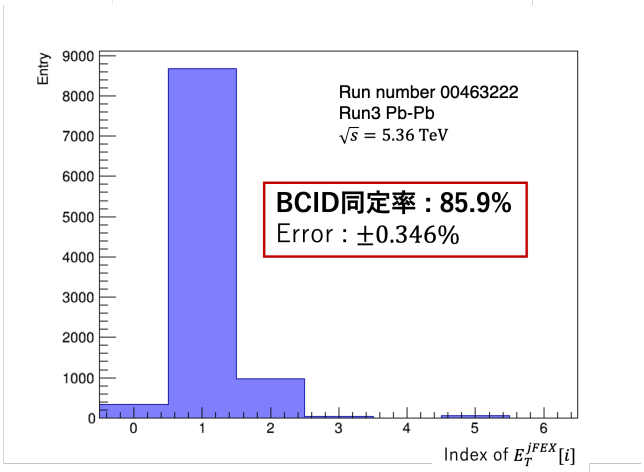
(a) *main* $E_T^{SC} \geq 0$ GeV の SC を含む jFEX の index 分布



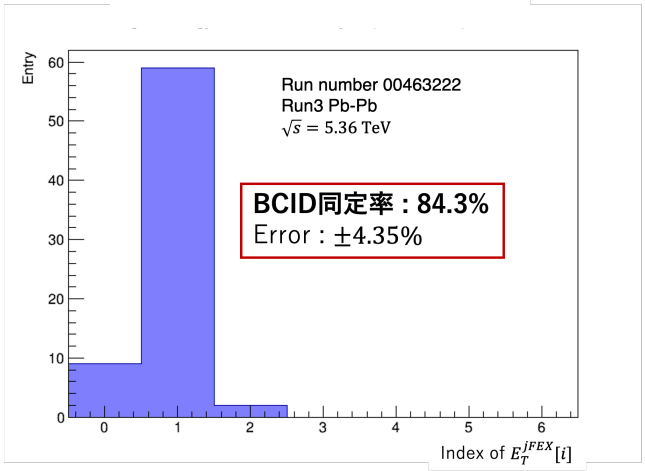
(b) *main* $E_T^{SC} \geq 0.4$ GeV の SC を含む jFEX の index 分布



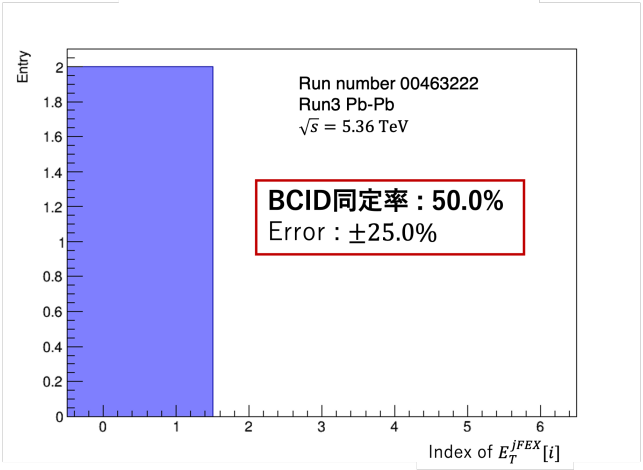
(c) *main* $E_T^{SC} \geq 0.8$ GeV の SC を含む jFEX の index 分布



(d) *main* $E_T^{SC} \geq 1$ GeV の SC を含む jFEX の index 分布



(e) *main* $E_T^{SC} \geq 5$ GeV の SC を含む jFEX の index 分布

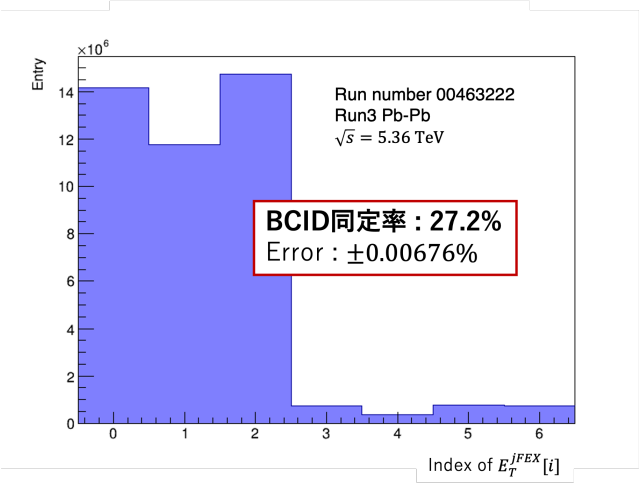


(f) *main* $E_T^{SC} \geq 10$ GeV の SC を含む jFEX の index 分布

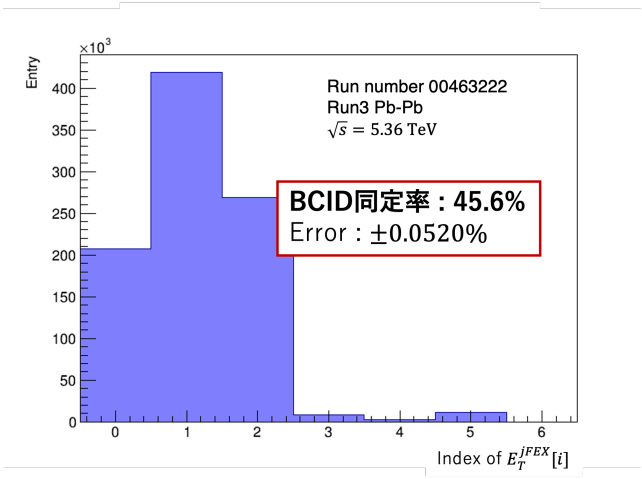
図 4.8: E_T^{jFEX} を最大にする index 分布 (HI run, *main* E_T^{SC} base)

さらに、jFEX のメイン読み出し横エネルギー *main* E_T^{jFEX} をベースにした index 分布も作った。上と同様に、*main* E_T^{jFEX} が 0, 0.4, 0.8, 1, 5, 10 GeV 以上の jFEX だけを抽出して、E_T^{jFEX} を最大にする index 分布を作った。重イオン衝突データについて、その結果を図 4.9 に示す。全体的な特徴は

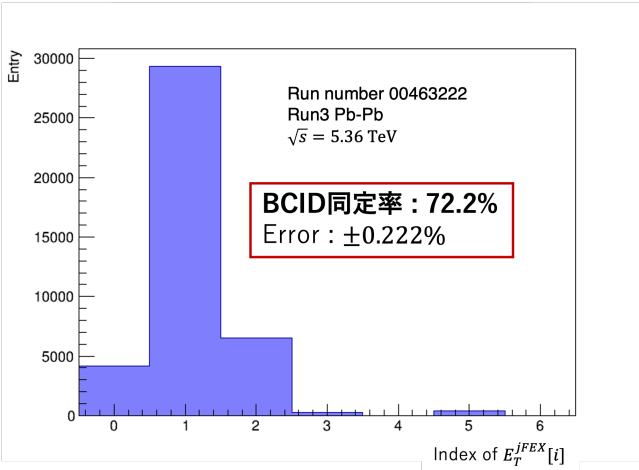
main E_T^{SC} ベースの index 分布とそれほど変わらない。*main* E_T^{jFEX} が 1 GeV 以上の jFEX の BCID 同定率は 80.6% (統計誤差は ±0.309%) であった。



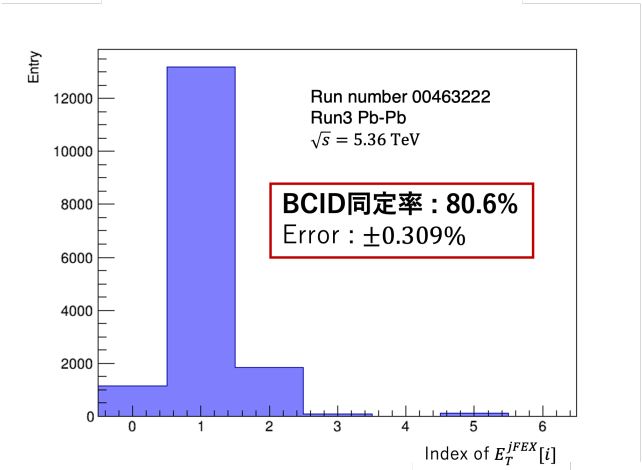
(a) *main* $E_T^{jFEX} \geq 0$ GeV の jFEX の index 分布



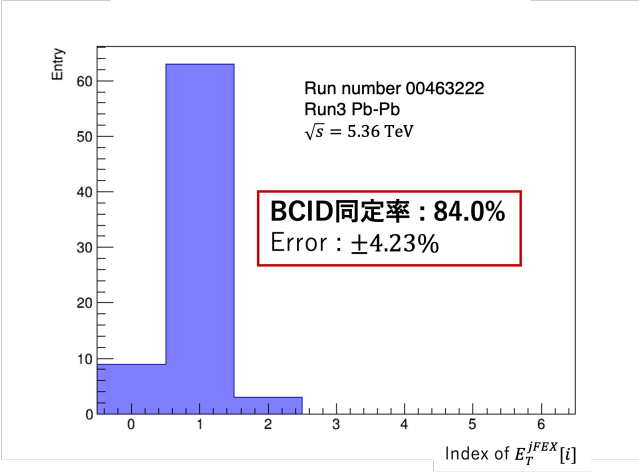
(b) *main* $E_T^{jFEX} \geq 0.4$ GeV の jFEX の index 分布



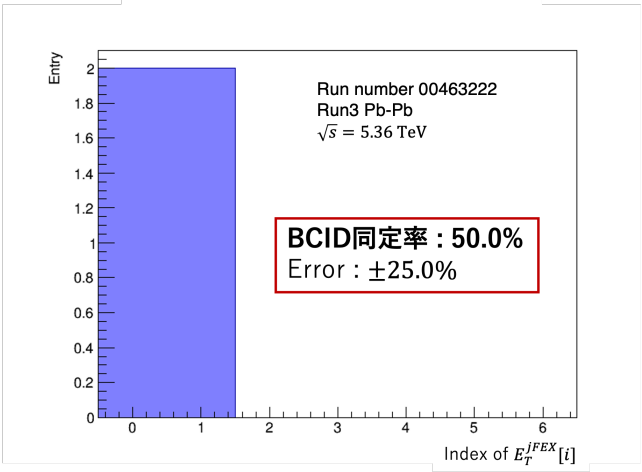
(c) *main* $E_T^{jFEX} \geq 0.8$ GeV の jFEX の index 分布



(d) *main* $E_T^{jFEX} \geq 1$ GeV の jFEX の index 分布



(e) *main* $E_T^{jFEX} \geq 5$ GeV の jFEX の index 分布



(f) *main* $E_T^{jFEX} \geq 10$ GeV の jFEX の index 分布

図 4.9: E_T^{jFEX} を最大にする index 分布 (HI run, *main* E_T^{jFEX} base)

また、UPC の物理で重要になるエネルギー領域 (2 GeV から 5 GeV) での index 分布も図 4.10 に示

す。この領域での BCID 同定率は 92.2% (統計誤差は ±0.774%) であった。この結果も、重イオン衝突実験においては Peak Finding Algorithm が有効である可能性を示唆している。

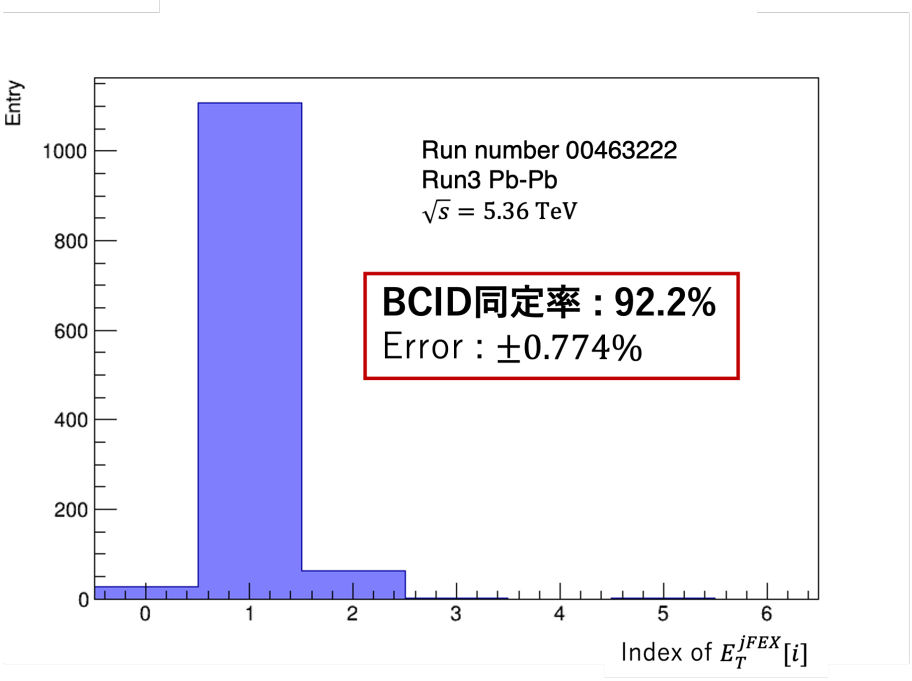
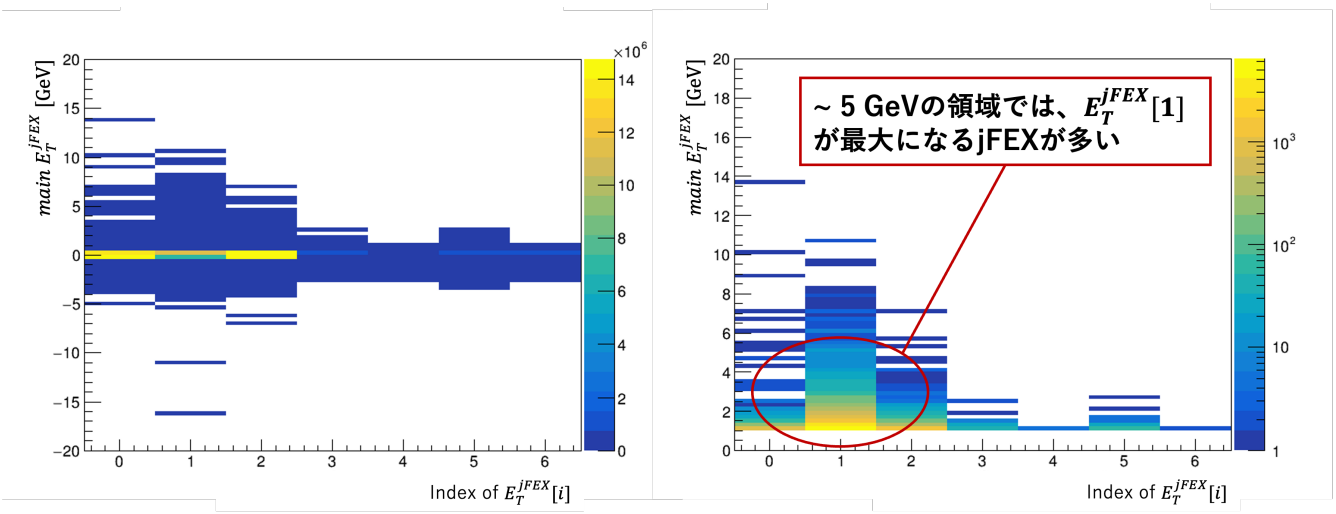


図 4.10: 2 GeV ≤ *main* E_T^{jFEX} < 5 GeV の jFEX の index 分布 (HI run, *main* E_T^{jFEX} base)

最後に、横軸が E_T^{jFEX} を最大にする index、縦軸がその jFEX の *main* E_T^{jFEX} の 2 次元ヒストグラムを図 4.11 に示す。図 4.11a は jFEX に対して特に何のエネルギーの条件も課していないもので、−20 GeV ≤ *main* E_T^{jFEX} < 20 GeV のエネルギー領域に渡ってその関係を可視化している。一方、図 4.11b は *main* E_T^{jFEX} が 1 GeV 以上の jFEX のみの分布であり、UPC で重要になる低いエネルギー領域では $E_T^{jFEX}[1]$ が最大になる jFEX が多く見られる。



(a) −20 GeV ≤ *main* E_T^{jFEX} < 20 GeV　　**(b)** 1 GeV ≤ *main* E_T^{jFEX}

図 4.11: E_T^{jFEX} を最大にする index と *main* E_T^{jFEX} の関係 (HI run)

4.3.2 *main* E_T^{jFEX} と $E_T^{PeakFinder}$ の相関

ここでは、*main* E_T^{jFEX} と Peak Finding Algorithm の出力である $E_T^{PeakFinder}$ の相関を見る。前章でも触れたが、以下に $E_T^{PeakFinder}$ の定義を示す。

$$E_T^{PeakFinder} = \begin{cases} E_T^{jFEX}[1], & \text{if } E_T^{jFEX}[1] \text{ is the peak} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

式 (4.1) で注意すべき点は 0 と出力されるケースである。本研究では簡単のため、$E_T^{jFEX}[1]$ が唯一のピークである場合のみ Peak Finding Algorithm の出力として採用する[12]。この条件は将来的に変わる可能性があるが、本研究ではこのまま検証を進めた。

図 4.12 に *main* E_T^{jFEX} と $E_T^{PeakFinder}$ の相関を示す。ただしここでは、*main* E_T^{jFEX} が正の領域のみ示してある。この図によると、*main* E_T^{jFEX} が 2 GeV 以上の領域では $E_T^{PeakFinder}$ との間に線形性が確認できる。これは言い換えれば、*main* E_T^{jFEX} と $E_T^{jFEX}[1]$ の線形関係を意味する。ただ、ある程度のエネルギーを持つ場合でも、数は少ないものの BCID の同定が正しくできていない jFEX の存在も確認できた[13]。

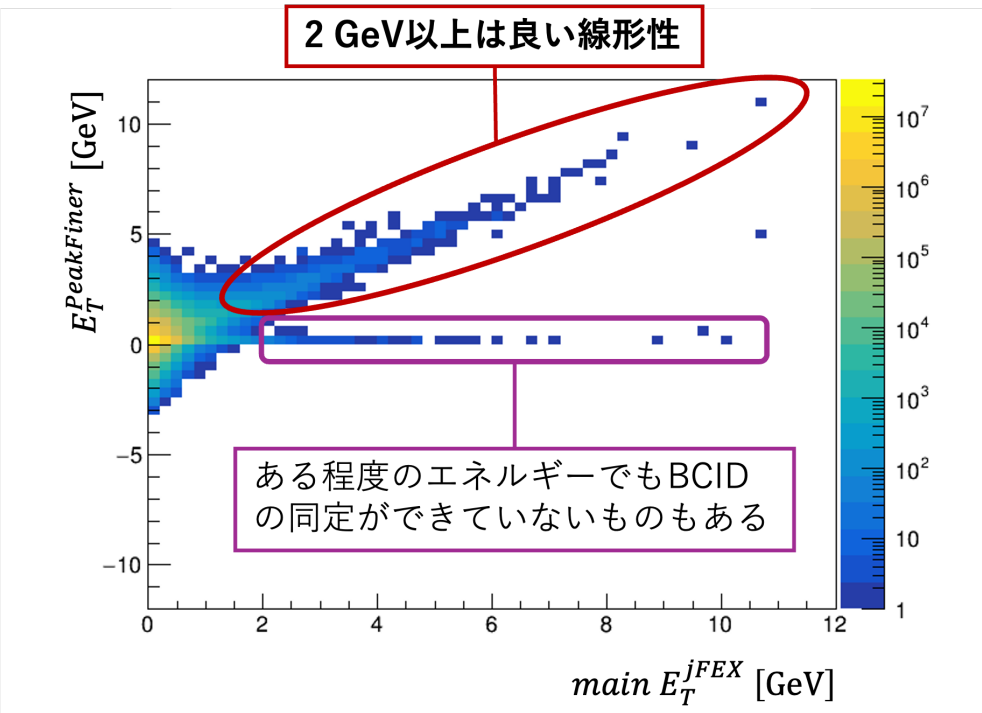


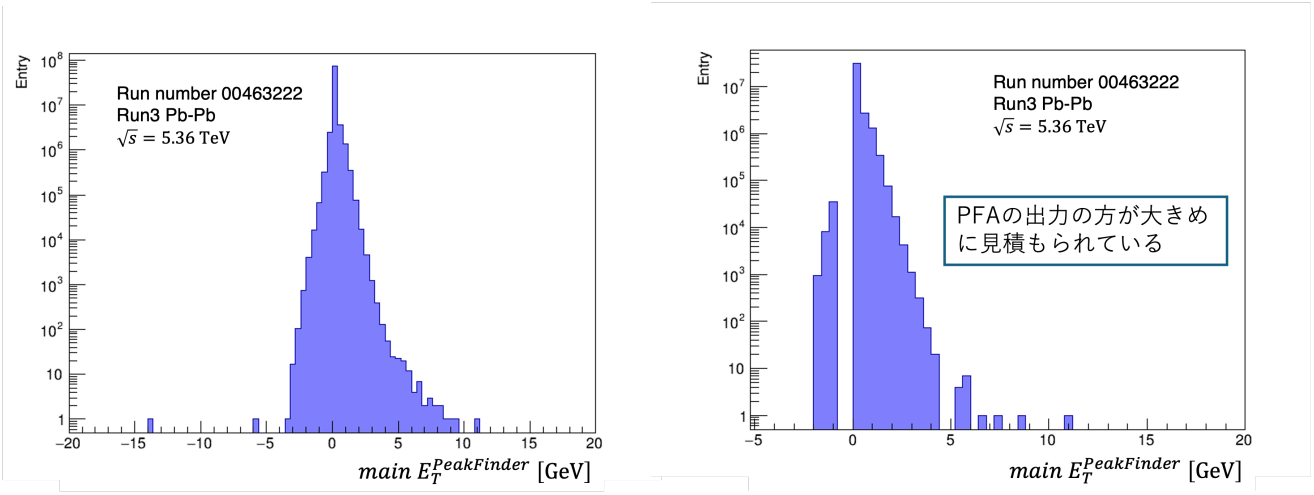
図 4.12: *main* E_T^{jFEX} と $E_T^{PeakFinder}$ の相関 (HI run)

この図は、Peak Finding Algorithm により正しい BCID が同定できた場合、jFEX のエネルギーが一定の精度で再構成されることを示している。この精度の見積りと評価については次節で詳しく述べる。

[12]前後の BC で全く同じ値を持つ場合など特殊なケースも起こり得るが、今は考えないことにする。
[13]pp run でも同様の図を作ったが、比較的高いエネルギー領域でも一定数 BCID の同定が正しくできていない jFEX があった。しかし、少数派であることに違いはない。

4.3.3　$E_T^{PeakFinder}$ の分布と $main\ E_T^{jFEX}$ との差分

ここでは、重イオン衝突データの解析から得られた $E_T^{PeakFinder}$ の分布と、$E_T^{PeakFinder}$ と $main\ E_T^{jFEX}$ の差について見る。図 4.13a は $E_T^{PeakFinder}$ の分布を示しており、図 4.12 の y 軸方向への射影に対応する。図 4.4a に示した、$main\ E_T^{jFEX}$ の分布をおおよそ再現しているのが分かる。また、図 4.13b は、図 4.13a の分布から図 4.4a の分布を引いた結果を示している。全体的に Peak Finding Algorithm の出力の方が jFEX のメイン読み出し横エネルギーよりも大きいことが確認できる。



(a) $E_T^{PeakFinder}$ の分布　　**(b)** $E_T^{PeakFinder}$ から $main\ E_T^{jFEX}$ を差し引いた分布

図 4.13: $E_T^{PeakFinder}$ の分布と $main\ E_T^{jFEX}$ との差分 (HI run)

4.4　Peak Finding Algorithm の評価

ここまで、重イオン衝突実験向けの液体アルゴンカロリメータデジタルトリガーへの実装が予定されている Peak Finder が持つアルゴリズム、すなわち Peak Finding Algorithm がどの程度有効であるか、2023 年の秋に取得された実データを用いた検証結果についてまとめた。ここでは、検証の最後のステップとして、Peak Finding Algorithm が正しい BCID を同定できた場合、どのくらいのエネルギーでこのアルゴリズムが有効だと言えるか、定量的な評価を行う。

4.4.1　評価式の定義

評価式の定義を式 (4.2) に示す。分子の E_T^{jFEX} [1] は正しい BCID を同定できたときの $E_T^{PeakFinder}$ の値 (式 (4.1) を参照) で、分母分子ともに現れる $main\ E_T^{jFEX}$ はこれまで何度も出てきたように jFEX のメイン読み出し横エネルギーである。ただし、分母を見て分かるように、偶然にも $main\ E_T^{jFEX}$ が 0 の jFEX に関してはこれ以降の評価計算に考慮しない[14]。

[14]表 4.3 で見たように、本研究で用いた重イオン衝突データから構築される jFEX は計 82,164,200 個であった。このうち、$main\ E_T^{jFEX} = 0$ となる jFEX は 22,693 個である。これは全体の 0.028% 程度であり、評価計算で無視してもほとんど問題ない。さらに言えば、そもそもほとんど 0 のエネルギーの jFEX を評価する意味はあまりない。

$$eval = \frac{E_T^{jFEX}[1] - main\ E_T^{jFEX}}{main\ E_T^{jFEX}} \tag{4.2}$$

例えば、Peak Finding Algorithm により正しく同定された BCID の jFEX のエネルギー $E_T^{jFEX}[1]$ と jFEX のメイン読み出し横エネルギー $main\ E_T^{jFEX}$ が等しい場合、分子は 0 になり、評価値 $eval$ は 0 になる。また、$E_T^{jFEX}[1]$ が $main\ E_T^{jFEX}$ の 2 倍の値を持つとき、評価値 $eval$ は 1 になる[15]。このように、高い精度で jFEX のエネルギーを再構成できれば、評価値 $eval$ の絶対値は 0 に近づく。以降、この評価式を使ってアルゴリズムの評価を行う。

4.4.2　評価値の jFEX メイン読み出し横エネルギー依存性

横軸 jFEX メイン読み出し横エネルギー、縦軸評価値 $eval$ の 2 次元ヒストグラムを図 4.14 に示す。これを見ると、Peak Finding Algorithm で BCID の同定が正しくできた場合、その jFEX のメイン読み出し横エネルギーが高ければ高いほど、再構成された jFEX のエネルギーはそれと近い値をとる、すなわち精度が高くなるのが確認できる。UPC の物理で重要になる 2 GeV 以上の領域では評価値の絶対値は 1 未満のものが大半を占める。一方、jFEX メイン読み出し横エネルギーが 1 GeV 未満の低いエネルギー領域では BCID の同定に成功しても、再構成された jFEX のエネルギーの精度が極端に悪いことが分かる。ただ、この領域は実際のトリガーでそれほど問題にならない領域のため、アルゴリズム自体が決して悪いわけではない。

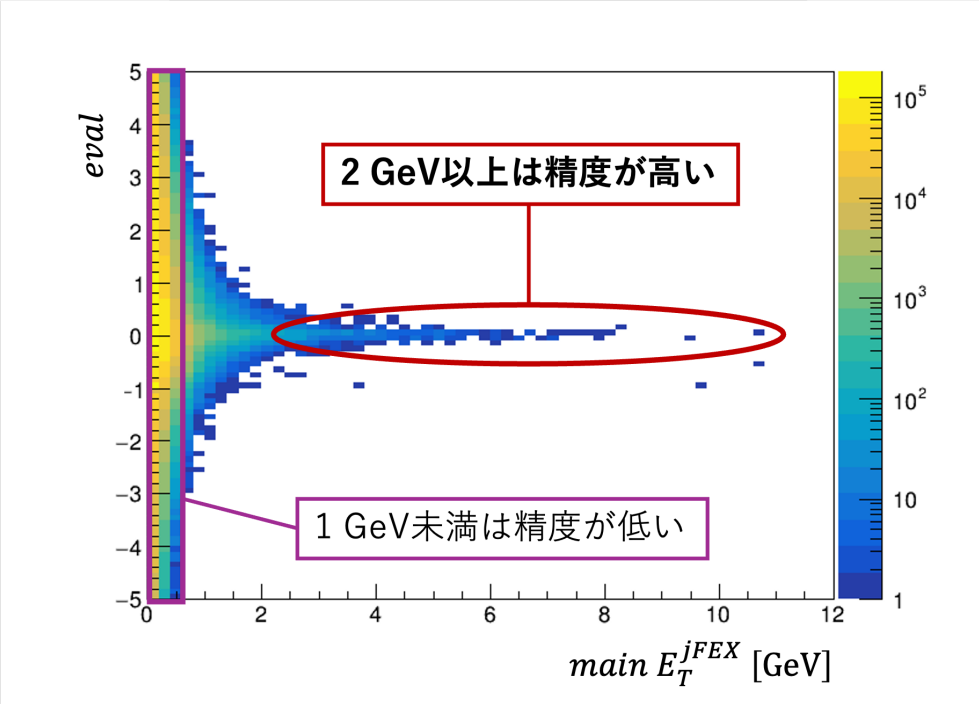


図 4.14: 評価値の jFEX メイン読み出し横エネルギー依存性 (HI run)

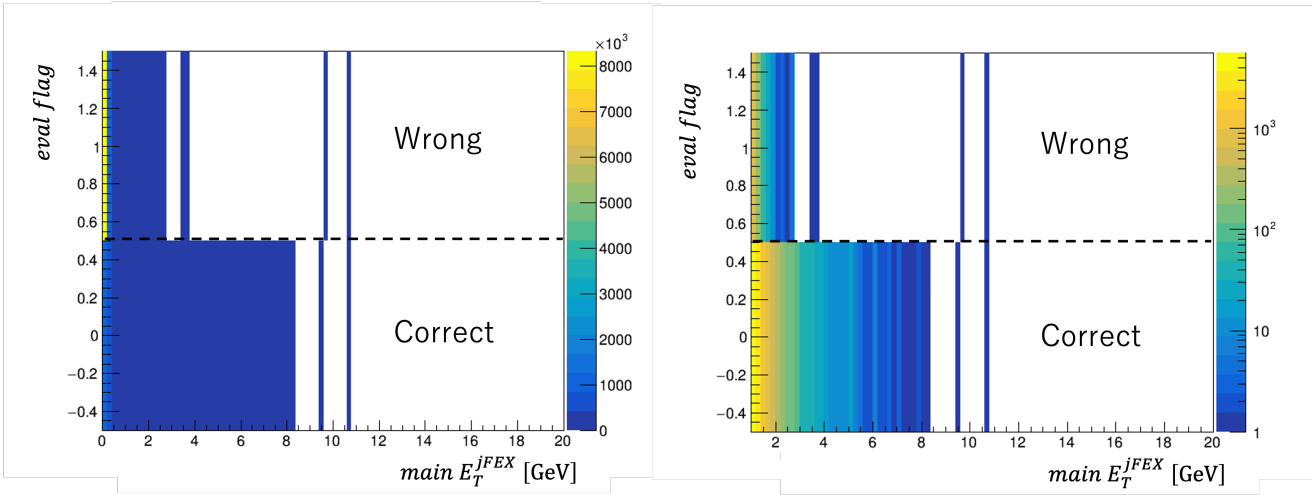
[15]評価値 $eval$ の値は $-\infty$ から $+\infty$ をとり得る。

4.4.3 jFEX メイン読み出し横エネルギーと評価フラグ

図 4.14 の 2 次元ヒストグラムをさらに詳しく分析するために、以下のような評価フラグを定義した。

$$eval\ flag = \begin{cases} 0\ (Correct), & \text{if } -0.5 \leq eval < 0.5 \\ 1\ (Wrong), & \text{otherwise} \end{cases} \tag{4.3}$$

式 (4.3) を、BCID が正しく同定できた全ての jFEX に適用し、図 4.15 に示すような、横軸 jFEX メイン読み出し横エネルギー、縦軸評価フラグの 2 次元ヒストグラムを作成した。図 4.15a は $main\ E_T^{jFEX}$ が 0 GeV 以上の範囲で、図 4.15b は 0 GeV 付近の jFEX を取り除く目的のため $main\ E_T^{jFEX}$ が 1 GeV 以上の範囲で描かれている。図 4.15a では見えないが、図 4.15b では割り当てられた評価フラグの数の違いが色の違いとして見えている。$main\ E_T^{jFEX}$ が 3 GeV 以上で、ほとんどの jFEX が $Correct$ に割り当てられている。次項ではこの図を利用して、BCID が正しく同定できた jFEX に対して、Peak Finding Algorithm の有効性を評価する。



(a) 0 GeV $\leq main\ E_T^{jFEX}$ **(b)** 1 GeV $\leq main\ E_T^{jFEX}$

図 4.15: jFEX メイン読み出し横エネルギーと評価フラグ (HI run)

4.4.4 Peak Finding Algorithm の有効性評価

図 4.15a について、横軸のビンあたりの $Correct\ flag$ の割合を計算したものが図 4.16 である。この図によると、$main\ E_T^{jFEX}$ が 0.4 GeV 以上であれば、BCID が正しく同定された jFEX のうち 50% 以上の jFEX が、$main\ E_T^{jFEX}$ とのずれが 50% 以内に収まる範囲で jFEX のエネルギーを再構成できることを意味する。同様に、$main\ E_T^{jFEX}$ が 1.2 GeV 以上であれば、BCID が正しく同定された jFEX のうち 90% 以上の jFEX が、$main\ E_T^{jFEX}$ とのずれが 50% 以内に収まる範囲で jFEX のエネルギーを再構成できることを意味する。

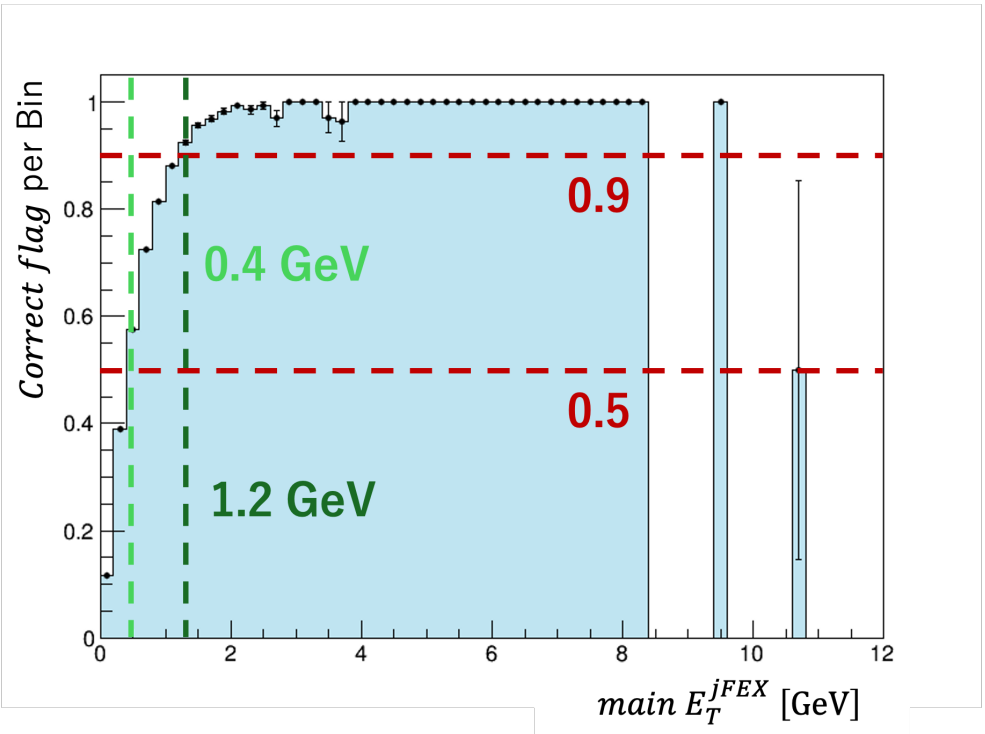


図 4.16: *Correct flag* に割り当てられる jFEX の割合 (HI run)

以下に、重イオン衝突実験データを用いた Peak Finding Algorithm の検証結果をまとめる。

- **BCID 同定率**：Peak Finding Algorithm の適用により、UPC の物理で重要になる 2 GeV から 5 GeV のエネルギー領域では、92.2% (統計誤差は ±0.774%) の jFEX が正しい BCID を同定できる (図 4.10)。
- **BCID が正しく同定された jFEX のエネルギーの精度評価**：*main* E_T^{jFEX} が 1.2 GeV 以上であれば、BCID が正しく同定された jFEX のうち 90% 以上の jFEX が、*main* E_T^{jFEX} とのずれが 50% 以内に収まる範囲で jFEX のエネルギーを再構成できる (図 4.16)。

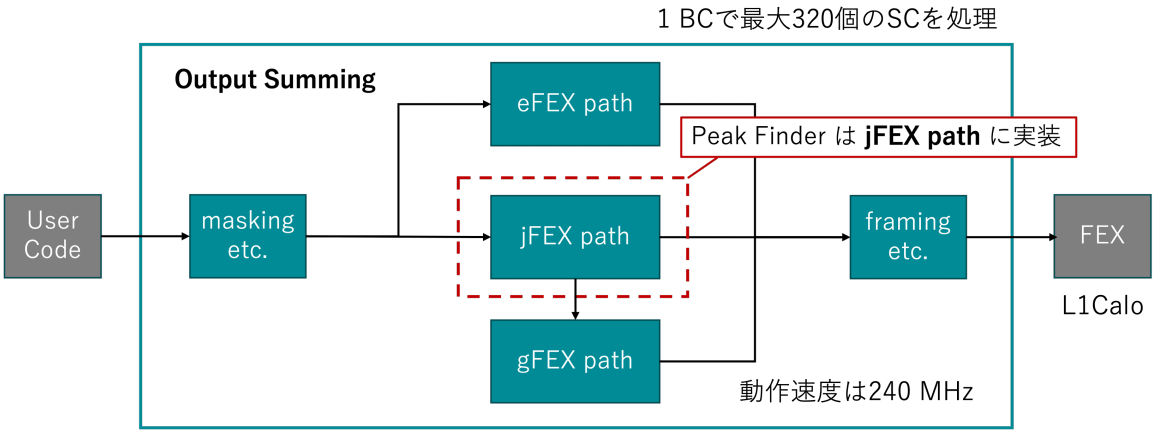
次章では、この結果を踏まえ、Peak Finding Algorithm を実装した Peak Finder の HLS による開発とシミュレーション結果について述べる。

第5章　HLSによるPeak Finderの開発

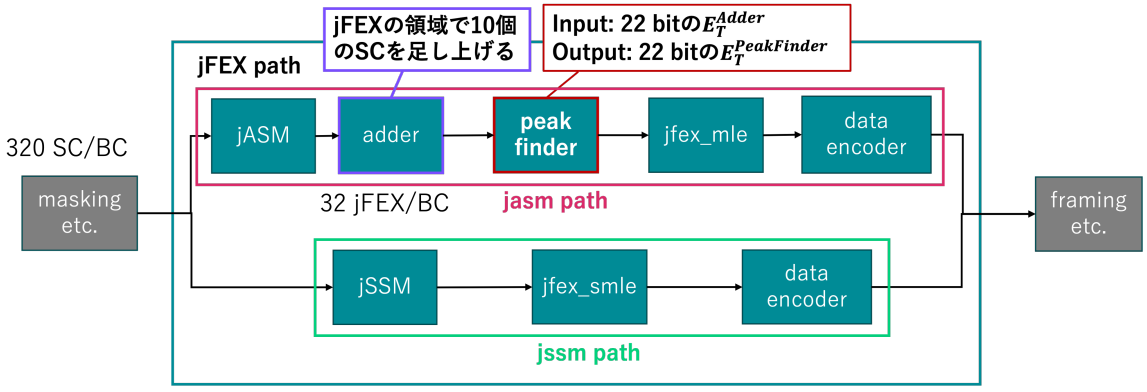
重イオン衝突向けのトリガーを改善させる目的で提案された Peak Finding Algorithm (PFA) は、Peak Finder という block 名で LATOME ファームウェアの Output Summing (Osum) に実装する予定である。通常、LATOME ファームウェアを実装する FPGA 内の処理は VHDL や Verilog といった Hardware Description Language (HDL) で記述されるが、本研究は C++言語による設計が可能な High-Level Synthesis (HLS) と呼ばれる新しい方法で LATOME ファームウェアへの実装を目指す。

LATOME ファームウェアへの実装に先立ち、まず Peak Finder 単体の機能設計と実装を行った。この章ではその内容とシミュレーション結果について述べる。

5.1　Output Summingにおける Peak Finder



(a) Osum の概要図



(b) Osum における Peak Finder

図 5.1: Osum の概要図と Peak Finder の位置

Peak Finder は、第 2 章の 2.4 節で説明したように、LATOME ファームウェアのデータを取りまとめて L1Calo の FEX に送る役割を果たす Output Summing (Osum) へ実装する。図 5.1 に Osum の概要図と Peak Finder の位置を示す。図 5.1a は Osum の概要図だが、いくつかの block は省略している。左側が User Code で、User Code からは Super Cell ごとに 18 bit のエネルギーが送られる。現在使われている LATOME ファームウェアでは、この時点ですでに正しいタイミング (BCID) で計算された Super Cell のエネルギーが送られてくるが、Peak Finder はそれ自身で BCID の同定を行う必要があるので、User Code の tau criteria を通る前のエネルギーを受け取る必要がある。以下では、これを前提として説明していく。

User Code から送られてくる 18 bit の Super Cell のエネルギーは Osum に入り、その信号が有効かどうかを判断するために masking 処理が行われる。さらに、EMEC 領域の一部の Super Cell は数の調整のため jFEX の領域で足し合わせが行われる。その後、eFEX path と jFEX path にデータが送られる。jFEX path は途中で gFEX path に経路が分岐し、それぞれの path で並行してデータが処理される。

eFEX path では Multi Linear Encoder (MLE) と呼ばれる方式で、Super Cell のエネルギーに応じて対応するコードを生成する。このコードは BCID とともにさらにエンコーディングされ、後段の framing を行う block に送られる。jFEX path では、図 5.1b に示すように、さらに 2 つの path (jasm path と jssm path) に分かれ、Peak Finder は jasm path に実装する。両者の違いは主に adder を含むかどうかであり、adder は jFEX の領域 (典型的には $\Delta\eta \times \Delta\phi = 0.1 \times 0.1$) で最大 10 個の Super Cell のエネルギーを足し上げる役割を担う。Osum では 1 BC で最大 320 個の Super Cell が処理されるので、adder では 32 個のデータにまとめられる。その出力は 22 bit の jFEX のエネルギーである。Peak Finder は peakfinder という block 名で adder の直後に実装する。したがって、peakfinder の入力は jFEX の領域ですでに足し上げられた 22 bit の jFEX のエネルギー E_T^{Adder} であり、出力は Peak Finding Algorithm が適用された 22 bit の jFEX のエネルギー $E_T^{PeakFinder}$ である。その後は、eFEX path と同様にコード情報にまとめられて framing を行う block にデータが送られる。gFEX path の処理も基本的には jFEX path と同じで、さらに広い領域 (典型的には $\Delta\eta \times \Delta\phi = 0.2 \times 0.2$) でエネルギーが足し上げられる。その後、コード情報にまとめられて framing を行う block にデータが送られる。

Osum に実装する Peak Finder のアルゴリズムは、簡単のため最もシンプルなものを採用した。それは現在の BC を i とすると以下のように書ける。式 (5.1) では、一つ前の BC のエネルギーが唯一のピークだった場合にのみその値を後段に送信し、それ以外は 0 を送信する。例えば、2 BC あるいは 3 BC に渡って全く同じ値で共にピークだった場合は 0 を割り当てて送信する。

$$E_T^{PeakFinder}[i] = \begin{cases} E_T^{Adder}[i-1], & \text{if } E_T^{Adder}[i-1] \text{ is the peak} \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

adder からは BC ごとに jFEX のエネルギーが送られてくるが、Peak Finding Algorithm の性質上、現在の BC で受け取ったエネルギーがピークかどうか判断するためには少なくとも 1 BC 待たなければならない。つまり、Peak Finder は少なくとも 1 BC のレイテンシを持つ。次節では、Peak Finder の開発を行った環境や使用したツールについて説明する。

5.2 開発環境とツール

Peak Finder の開発は CERN が提供する HLS 向けの machine (OS は Alma 9) で行った。GUI でシミュレーションの様子を確認する際はリモートデスクトップを使った。また、現地の HLS 開発グループから Osum が実装されたコードを受け取り、Git でバージョンを管理しながら開発を進めた[1]。以下に、Peak Finder の開発で使ったツールを並べる。

- **Visual Studio Code**：ターミナルから CERN の machine に ssh 接続が可能で、ファイルの管理や C++によるコード開発が可能。Microsoft 社提供。Version は 1.85.2。
- **Eclipse**：Visual Studio Code で開発したテストベンチを使って、C++シミューレションの実行が可能。デバッグ機能が豊富で、block ごとにシミューレションしたり Osum 全体でシミュレーションすることもできる。非営利組織 Eclipse Foundation 提供。Version は 2022-06 (4.24.0)。
- **Catapult**：C++で記述された block に対して HLS を施行し、Verilog や VHDL に変換する。クロックの設定や block の Architecture (ループ処理の並列化やパイプラインの指定など) を GUI で設定可能で、設定した FPGA のリソース使用量や遅延時間を最適化するように RTL 生成を行う。Siemens 社提供。Version は 2023.2_1/1065141 (Production Release)。
- **QuestaSim**：Catapult で RTL 生成した block と C++で記述されたテストベンチを用いて、RTL シミュレーションを行う。データが転送されるタイミングが波形として可視化され、タイミング違反やエラーの確認ができる。Siemens 社提供。Version は 2021.3_2。

5.3 C++シミュレーション

5.3.1 テストベンチの設計と構造

本研究で Osum に実装する Peak Finder の構造を図 5.2 に示す。一番外側は Osum 内の環境を模したもので、peakfinder_top という名前のトップレベルの関数である。adder からは BC ごとに 32 個の jFEX に関するエネルギーが入ってくるため、トップレベルの中にはその値を 3 BC に渡って保存するための 32×3 のバッファ mem[32][3] と、32 回の loop によって展開された peakfinder という関数がある。バッファは adder からデータを受け取ると、mem[ ][0] の内容を mem[ ][1] に、mem[ ][1] の内容を mem[ ][2] にシフトしてから、mem[ ][0] に新しいデータを順に格納する。その後、jFEX i に関連する 3 BC 分のエネルギー mem[i][0]、mem[i][1]、mem[i][2] をループ番号 i の peakfinder に渡す。peakfinder は式 (5.1) に従ってピークの判断を行い、mem[i][1] の内容または 0 を出力する。peakfinder は 1 BC で 32 回呼び出されるので出力のサイズは 32 になる。また、動作速度は 240 MHz である。

[1]HLS 開発グループから受け取った Osum のバージョンは gFEX path がまだ実装されていなかった。したがって、本研究では eFEX path と jFEX path のみが実装された Osum でシミュレーションを行った。

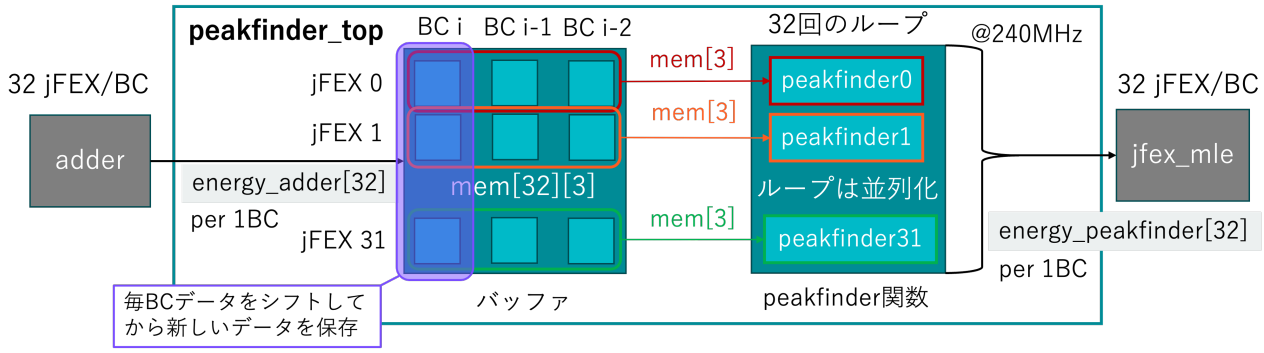


図 5.2: Peak Finder トップレベルの構造

図 5.3 はトップレベルの C++コードの一部を示している。SHIFT_LOOP はバッファのシフティングに関するループで、peakfinder_loop は peakfinder 関数を複数回呼び出すためのループである。また、15 行目と 22 行目に書かれている #pragma unroll yes は、HLS 特有のコメントで、ループを逐次実行ではなく並列で実行するよう合成したいときにつける。このコメントは C++シミュレーションでは無視されるが、HLS を実行するときに有効になる。

```
13      static data_22b_signed mem[NUM_STREAMS/SIZE_ADDER][3];  // Buffer to save 32
14
15      #pragma unroll yes
16      SHIFT_LOOP : for (unsigned int i = 0; i < NUM_STREAMS / SIZE_ADDER; i++) {
17          mem[i][2] = mem[i][1];
18          mem[i][1] = mem[i][0];
19          mem[i][0] = energy_adder[i];    // mem[i][0] is always new
20      }
21
22      #pragma unroll  // peakfinder
23      peakfinder_loop : for (unsigned o = 0; o < NUM_STREAMS / SIZE_ADDER; ++o) {
24          peakfinder(mem[o], energy_peakfinder[o]);
25      }
```

図 5.3: peakfinder_top の内容

上の peakfinder_top 関数が期待通りの動作をするかテストするためのテストベンチ peakfinder_top_tb 関数を設計した。そのおおよその流れを以下に述べる。peakfinder_top 関数は 1 回の呼び出しで、32 個の jFEX のエネルギー $E_T^{Adder}[32]$ を受け取り、それぞれの jFEX に対して peakfinder 関数を呼び出す。その後、要素数 32 のデータ $E_T^{PeakFinder}[32]$ を出力する。

[テストベンチの構造]

1. 指定した数だけ 22 bit の入力データ E_T^{Adder} を生成する。
2. 入力データ E_T^{Adder} を CSV ファイルに保存する。
3. 生成した入力から期待される出力 $E_T^{PeakFinder\_EXP}$ を計算し、vector に格納する。
4. 指定した BC の数だけ peakfinder_top 関数を呼び出す。
5. peakfinder_top 関数の出力 $E_T^{PeakFinder}$ と $E_T^{PeakFinder\_EXP}$ を比較し、エラーの確認をする。

6. peakfinder_top 関数の出力 $E_T^{PeakFinder}$ を CSV ファイルに保存する。

7. Summary を表示し、シミュレーションを終了する。

これらは C++で記述し、シミュレーションには Eclipse を用いた。C++シミュレーションは、ランダム入力を用いた場合と 729 通りの主要な入力を用いた場合で行った。

5.3.2 ランダム入力を用いたテスト

ここではランダムな入力に対する C++シミュレーション結果について述べる。簡単のため、入力データの範囲は 0 から 10 までの整数とし、10 BC 分でシミュレーションした。生成される入力データは 32 jFEX × 10 BC = 320 個である。図 5.4 にシミュレーション結果を示す。図 5.4a と図 5.4b はテストベンチが生成した peakfinder_top 関数の入力と出力に関する CSV ファイルで、図 5.4c は Eclipse でテストベンチを実行した時の出力である。図 5.4c にある通り、シミュレーションは成功した。その結果を確認するための例として、図 5.4b の jFEX 1 の BC 4 の値 (10) を見ると、これは図 5.4a の jFEX 1 の BC 3 の値 (10) が前後の BC と比較して最大であったため、この値が 1 BC 遅れて出力されているのが分かる。

			jFEX 0,	jFEX 1,	jFEX 2,	jFEX 3,	jFEX 4,	jFEX 5,	jFEX 6,
2	BC	0,	9,	3,	9,	5,	2,	10,	1,
3	BC	1,	1,	9,	5,	8,	9,	10,	7,
4	BC	2,	4,	5,	6,	2,	2,	6,	0,
5	BC	3,	6,	10,	9,	1,	9,	9,	6,
6	BC	4,	9,	2,	0,	10,	5,	7,	8,
7	BC	5,	9,	9,	8,	3,	6,	4,	6,
8	BC	6,	1,	4,	5,	4,	4,	0,	1,
9	BC	7,	6,	1,	4,	6,	2,	3,	9,
10	BC	8,	10,	5,	0,	6,	0,	6,	0,
11	BC	9,	0,	0,	0,	2,	2,	6,	5,
12									

(a) 生成された 10 BC 分のランダム入力データ (一部)

			jFEX 0,	jFEX 1,	jFEX 2,	jFEX 3,	jFEX 4,	jFEX 5,	jFEX 6,
2	BC	0,	0,	0,	0,	0,	0,	0,	0,
3	BC	1,	9,	0,	9,	0,	0,	0,	0,
4	BC	2,	0,	9,	0,	8,	9,	0,	7,
5	BC	3,	0,	0,	0,	0,	0,	0,	0,
6	BC	4,	0,	10,	9,	0,	9,	9,	0,
7	BC	5,	0,	0,	0,	10,	0,	0,	8,
8	BC	6,	0,	9,	8,	0,	6,	0,	0,
9	BC	7,	0,	0,	0,	0,	0,	0,	0,
10	BC	8,	0,	0,	0,	0,	0,	0,	9,
11	BC	9,	10,	5,	0,	0,	0,	0,	0,
12									

(b) ランダム入力データに対する peakfinder_top 関数の出力 (一部)

```
Starting simulation.
Testing with 10 BCs.

Generating 32 * 10BC random signed integers and pack them into energy_adder_vec.
Saving energy_adder_vec to CSV file.

Computing the expected output of peakfinder_top().

Calling the function peakfinder_top().

Checking the result...
Skipping the first 64 output, which are meaningless.

Saving energy_peakfinder_vec to csv file.

TESTBENCH SUCCEEDED!!
error_count is 0

Summary
Number of BCs : 10
energy_adder_vec.size() = 320
expected_output.size() = 320
energy_peakfinder_vec.size() = 320

Total simulation time: 0.000760917 seconds.
../src/peakfinder_top_tb.cpp:259 - End of testbench.
```

(c) テストベンチからの出力

図 5.4: ランダム入力データに対するシミュレーション結果

5.3.3　主要な入力組み合わせを考慮したテスト

ランダム入力データによるシミュレーションでは 0 から 10 までの整数データしか扱っていなかったが、実際のデータはもっと広い値をとり得る。そこで、表 5.1 のように各 BC で 9 通りの値を仮定し、想定されるあらゆる組み合わせを持つ入力データでシミュレーションを行った。図 5.5 はその組み合わせの例である。このテストケースは $9 \times 9 \times 9 = 729$ 通り存在し、全て同じ値をとる場合、正負にまたがる場合、考えられる最大値や最小値を含む場合などが考慮されている。テストベンチではこれら 729 通りの組み合わせを生成し、peakfinder_top 関数の入力として渡した。また、一回の BC で peakfinder 関数が 32 回呼び出されるので、効率的にテストを行えるように 32 通りずつテストを行った[2]。

表 5.1: 主要な入力組み合わせ

Test Case	値
Maximum	2,097,151 $(2^{21}-1)$
Positive high	100,000
Positive middle	1,000
Positive low	1
Zero	0
Negative high	-1
Negative middle	-1,000
Negative low	-100,000
Minimum	-2,097,152 (-2^{21})

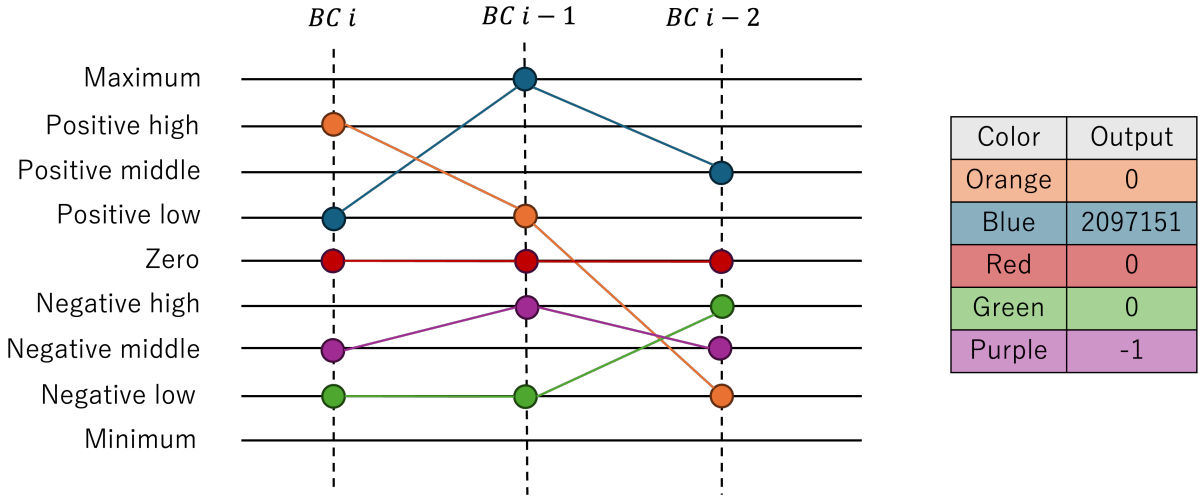


図 5.5: peakfinder_top 関数に入力される主要な組み合わせの例

図 5.6 にシミュレーション結果を示す。

[2] 1 通りにつき、3 BC 分のデータが必要なので 729 通りであれば、2,187 BC 分のデータが必要になる。しかし、一度に 32 通り処理できるので 69 BC 分データを用意できれば全ての組み合わせがテスト可能である。したがって、peakfinder_top 関数は 69 回呼び出される。

			jFEX 0,	jFEX 1,	jFEX 2,	jFEX 3,	jFEX 4,	jFEX 5,	jFEX 6,
2	BC	0,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,
3	BC	1,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,
4	BC	2,	2097151,	1000000,	1000,	1,	0,	-1,	-1000,
5	BC	3,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,
6	BC	4,	1,	1,	1,	1,	0,	0,	0,
7	BC	5,	-1,	-1000,	-100000,	-2097152,	2097151,	1000000,	1000,
8	BC	6,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,
9	BC	7,	-100000,	-100000,	-100000,	-100000,	-100000,	-100000,	-100000,
10	BC	8,	1000000,	1000,	1,	0,	-1,	-1000,	-100000,

(a) テストケースをもとに生成された 729 通りの入力 (一部)。各 jFEX は 3 BC で 1 つのパターンをテストする。

			jFEX 0,	jFEX 1,	jFEX 2,	jFEX 3,	jFEX 4,	jFEX 5,	jFEX 6,
2	BC	0,	0,	0,	0,	0,	0,	0,	0,
3	BC	1,	0,	0,	0,	0,	0,	0,	0,
4	BC	2,	0,	0,	0,	0,	0,	0,	0,
5	BC	3,	0,	0,	0,	0,	0,	0,	0,
6	BC	4,	0,	2097151,	2097151,	2097151,	2097151,	2097151,	2097151,
7	BC	5,	0,	0,	0,	0,	0,	0,	0,
8	BC	6,	0,	0,	0,	0,	0,	0,	0,
9	BC	7,	2097151,	2097151,	2097151,	2097151,	0,	2097151,	2097151,
10	BC	8,	0,	0,	0,	0,	0,	0,	0,

(b) 729 通りのテストケースに対する peakfinder_top 関数の出力 (一部)

```
Starting simulation.
Simulating with the all possible combinations.
Testing with 69 BCs.

Maximum value = 2097151
Positive high value = 1000000
Positive middle value = 1000
Positive low value = 1
Zero value = 0
Negative high value = -1
Negative middle value = -1000
Negative low value = -100000
Minimum value = -2097152

Generating input data.

Saving energy_adder_vec to CSV file.

Computing the expected output of peakfinder_top().

Calling the function peakfinder_top().

Checking the result...
Skipping the first 64 output, which are meaningless.

Saving energy_peakfinder_vec to csv file.

TESTBENCH SUCCEEDED!!
error_count is 0

Summary
Number of BCs : 69
energy_adder_vec.size() = 2208
expected_output.size() = 2208
energy_peakfinder_vec.size() = 2208

Total simulation time: 0.00267049 seconds.
../src/peakfinder_top_tb.cpp:259 - End of testbench.
```

(c) テストベンチからの出力

図 5.6: 729 通りのテストケースに対するシミュレーション結果

図 5.6a と図 5.6b はテストベンチが生成した peakfinder_top 関数の入力と出力が保存された CSV ファイルで、図 5.6c は Eclipse でテストベンチを実行した時の出力である。それぞれの図から読み取れるように、シミュレーションは成功した。この場合もやはり、1 BC 遅れてピークの判断が行われている。

以上、ランダム入力データと 729 通りの入力データを用いて peakfinder_top 関数のロジックが正しいことを確認した。次のステップとして、この関数に HLS を施行して RTL 生成を行った。次節ではその内容と結果について述べる。

5.4 HLS による RTL 生成

5.4.1 HLS と Catapult による RTL 生成の手順

High-Level Synthesis (HLS) は、C++などの高級プログラミング言語を使用してハードウェア動作を記述し、これを FPGA の RTL (Register Transfer Level) コードに自動変換する合成手法である。HLS はハードウェア設計プロセスを効率化する。具体的には、従来の RTL 設計と比べて抽象度が高いため設計者はアルゴリズムやシステムレベルの動作に集中でき、開発期間の短縮や設計の保守性の向上が可能となる。本研究では、Siemens 社が提供する HLS ツールである Catapult を用いて Peak Finder の開発を行った。以下に、Catapult による HLS の手順を説明する。

[Catapult による HLS 施行のステップ]

1. **Input Files**：HLS を実行する対象のファイルを指定する。ヘッダーファイルは自動で読み込まれる。本研究では、peakfinder_top.cpp、peakfinder.cpp、peakfinder_top_tb.cpp を指定した。ただし、3 つ目のテストベンチファイルは HLS の対象から除外するよう指定が必要である。

2. **Hierarchy**：Input されたファイルを基に、設計の階層構造を定義する。Catapult では、この階層を活用して設計の分割やリソース共有、最適化が行える。本研究では、peakfinder_top.cpp がトップレベルで、peakfinder.cpp が sub block である。

3. **Libraries**：合成の対象となる FPGA の設定や合成時に使用する標準ライブラリを指定する。本研究では、LATOME ファームウェアに搭載されている FPGA と同じ型番を指定した。

   [FPGA] Altera Arria 10 10AX115R3F40E2SG

   [ライブラリ] Altera_DIST, Altera_M20K, Altera_MLAB

4. **Mapping**：ライブラリをロードし、クロックの設定を行う。本研究では、240 MHz に設定した。

5. **Architecture**：パイプライン化、ループの並列化、共有メモリの設定などハードウェアアーキテクチャの設計を行い、リソースやレイテンシなどの性能要件を満たす構造を設計する。HLS による設計で最も重要なステップである。詳細は次項で述べる。

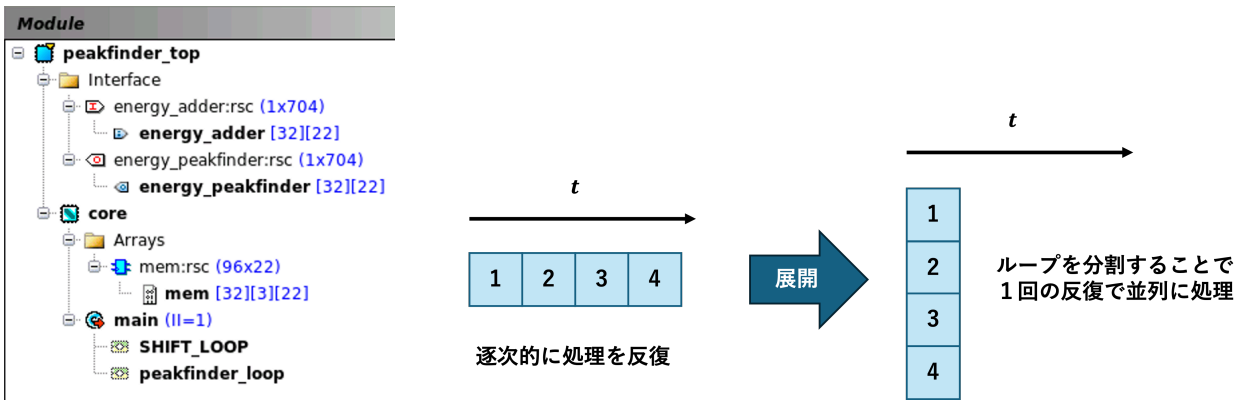
6. **Resources**：設計に使用するリソース（加算器、乗算器、メモリなど）の種類や数を分析し、必要に応じてリソース使用量の最適化を行う。

7. **Schedule**：設計全体のタイミングを決定する。具体的には、各操作の開始タイミングやパイプラインステージをスケジューリングし[3]、性能要件を満たすように調整を行う。

8. **RTL**：Input された C++コードと設計した Architecture を基に、FPGA の RTL (Verilog や VHDL) コードを生成する。この RTL は論理合成やシミュレーションに使用される。

5.4.2　合成の Architecture

Catapult で指定した合成の Architecture を図 5.7a に示す。これは peakfinder_top 関数の内容であり、Interface と core の 2 つのフォルダが確認できる。Interface には peakfinder block の入力及び出力のデータ形式が表示され、energy_adder[32] が adder からの入力 $E_T^{Adder}[32]$ で、energy_peakfinder[32] が jfex_mle への出力 $E_T^{PeakFinder}[32]$ である。それぞれ 22 bit のデータ幅を持つので、全体の幅は $22 \times 32 = 704$ bit である。一方、core の下にはメモリの仕様を管理する Arrays フォルダと peakfinder_top 関数の main 関数の構造を表す main フォルダが確認できる。5.3.1 項で見たように、3 BC 分のエネルギーを一時的に保存するためのバッファのメモリサイズもここに表示される。要素あたり 22 bit なので、そのサイズは $22 \times 32 \times 3 = 2,112$ bit である。main 関数は SHIFT_LOOP と peakfinder_loop という 2 つのループを持ち、そのどちらも完全に展開されている。



(a) peakfinder_top 関数の Architecture　　**(b)** 反復回数 4 のループの展開例

図 5.7: peakfinder_top 関数の Architecture とループの展開例

図 5.7b は、反復回数が 4 の場合のループの完全な展開についての例を表している。通常、C++などの高級言語で書かれたプログラムはソースコードの上から順に順次実行される。これを逐次処理という。しかし、FPGA などのハードウェア上では必ずしも上から順に実行する必要はない。各反復の間に値の依存関係が無い場合、異なるハードウェア資源を使って並列に処理することが可能である。これを可能にするための操作が「ループを展開する」である。本研究で実装したバッファ内のシフト操作などは、1 クロックの間に同時に実行した方がレイテンシの短縮に繋がり効率が良いので、このような Architecture 設計を行った。また、図には載っていないが、sub block の peakfinder 関数を順序回路として実装するの

[3]操作をいくつかの段階に分け、それぞれを時間軸上に順番に並べながら、複数の操作が重ならないように配置すること。これにより、一度に複数のデータを並列に処理できるようになる。

か組み合わせ回路として実装するのかなどの指定も可能である[4]。peakfinder 関数は 3 つのエネルギーを受け取り次第、レジスタを介すことなく瞬時に条件判断でピークを判断するので、組み合わせ回路として実装した。

5.4.3　RTL 生成結果

Architecture の設計後は RTL 生成を実行した。図 5.8 はその生成結果を示している。RTL 生成は peakfinder_top と peakfinder それぞれに対して行われ、リソース使用量なども個別に確認できる。ただし、peakfinder_top のリソースには peakfinder のリソースが含まれている。

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
peakfinder.v1 (extract)	0	0.00	0	0.00	2.40	47.06
peakfinder_top.v1 (extract)	**1**	**4.17**	**1**	**4.17**	**2.07**	**1514.89**

図 5.8: peakfinder_top 関数の HLS による RTL 生成結果

図中の各列の定義は以下の通りである。

- **Latency Cycles**：1 つの入力についての処理にかかるクロック数。
- **Latency Time**：Latency Cycles に対応する時間。単位は ns。
- **Throughput Cycles**：関数（ひとまとまりの処理）を一回呼び出すのに必要な最小のクロック数。
- **Throughput Time**：Throughput Cycles に対応する時間。単位は ns。
- **Slack**：指定したクロックサイクル時間のうち、ロジックや信号の伝搬に使用されなかった部分。単位は ns。負の値はタイミング違反を意味する。
- **Total Area**：設定した FPGA の構成単位を用いたリソースの推定使用量。

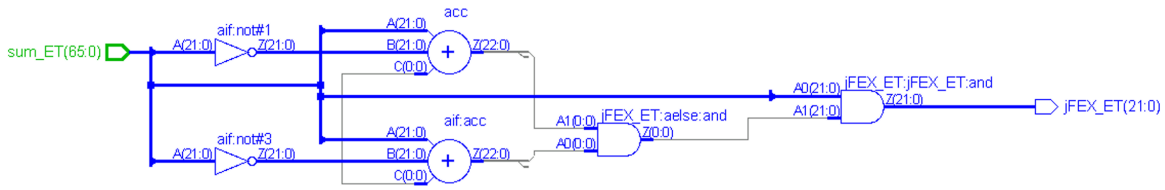
この図によると、peakfinder 関数はレイテンシとスループットがともに 0 である。これは単に、peakfinder 関数が組み合わせ回路として実装されたからだと考えられる。Slack は 2.40 ns なので、処理速度には余裕がある。また、リソース推定使用量は 47.06 であった。その内訳は図 5.9a にある通り、Area が 12.030 の加算器が 2 つ、Area が 1.000 の AND 回路が 1 つ、Area が 22.000 の AND 回路が 1 つ、Area が 0 の NOT 回路が 2 つである。また、図 5.9b にそれらの素子を含む peakfinder 関数の回路図を示す。図 5.9c は生成された Verilog コード (一部) であり、34 行目が peakfinder 関数の出力に該当する。

[4]組み合わせ回路は、入力信号の状態に基づいて即座に出力が決定する回路である。状態を記憶せず、過去の入力に依存しない。一方、順序回路は、入力信号とともに現在の状態（内部記憶）に基づいて出力を決定する回路である。過去の入力や状態を記憶するため、クロック信号に同期して動作する。

```
Bill Of Materials (Datapath)
  Component Name                  Area Score Area(DSP) Area(LUTs) Delay Post Alloc Post Assign
  ------------------------------- ---------- --------- ---------- ----- ---------- -----------
  [Lib: mgc_Altera-Arria-10-2_beh]
  mgc_add(22,1,22,1,23)               12.030     0.000     12.030 1.281          0           2
  mgc_and(1,2)                         1.000     0.000      1.000 0.260          0           1
  mgc_and(22,2)                       22.000     0.000     22.000 0.260          0           1
  mgc_not(22)                          0.000     0.000      0.000 0.000          0           2

  TOTAL AREA (After Assignment):      47.059     0.000     47.000
```

(a) peakfinder 関数のリソース推定使用量の内訳。Post Assign は割り当てられた論理素子の数を表す。



(b) peakfinder 関数の回路図。左側の sum_ET は 22 bit×3 BC のエネルギーで peakfinder 関数の入力である。右の jFEX_ET は 22 bit の出力を表す。

```
18  input [65:0] sum_ET;
19  output [21:0] jFEX_ET;
20
21
22  wire jFEX_ET_aelse_and_nl;
23  wire[22:0] aif_acc_nl;
24  wire[23:0] nl_aif_acc_nl;
25  wire[22:0] acc_nl;
26  wire[23:0] nl_acc_nl;
27
28  // Interconnect Declarations for Component Instantiations
29  assign nl_aif_acc_nl = conv_s2u_22_23(sum_ET[65:44]) - conv_s2u_22_23(sum_ET[43:22]);
30  assign aif_acc_nl = nl_aif_acc_nl[22:0];
31  assign nl_acc_nl = conv_s2u_22_23(sum_ET[21:0]) - conv_s2u_22_23(sum_ET[43:22]);
32  assign acc_nl = nl_acc_nl[22:0];
33  assign jFEX_ET_aelse_and_nl = (readslicef_23_1_22(aif_acc_nl)) & (readslicef_23_1_22(acc_nl));
34  assign jFEX_ET = MUX_v_22_2_2(22'b0000000000000000000000, (sum_ET[43:22]), jFEX_ET_aelse_and_nl);
```

(c) 生成された peakfinder 関数の Verilog のコード (一部)

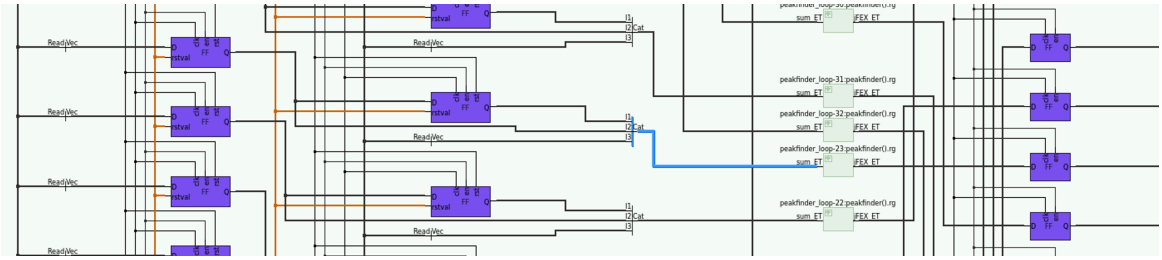
図 5.9: RTL 生成後の peakfinder 関数のリソース推定使用量、回路図、Verilog のコード

同様に、peakfinder_top 関数の RTL 生成結果についても述べる。図 5.8 では、レイテンシとスループットは 1 クロックサイクルであった。これは処理時間と関数の呼び出し間隔が共に 4.17 ns で実装可能であることを意味している。Slack は 2.07 ns で、これも処理時間に余裕があった。リソース推定使用量は 1514.89 であった。その内訳を図 5.10a に示す。これによると、一つあたりの Area が 47.059 の peakfinder 関数が 32 個、Area が 1.000 の AND 回路が 5 個、Area が 1.000 の NOR 回路が 2 個、Area が 1.000 の OR 回路 1 個、Area が 0 の NOT 回路やレジスタなどが複数含まれている。また、図 5.10b に peakfinder_top 関数の回路図の一部を示す。いくつかのレジスタと複製された peakfinder 関数が確認できる。図 5.10c は生成された Verilog コード (一部) であり、メモリのシフティングに関する処理が記述されている。

```
Bill Of Materials (Datapath)
  Component Name                    Area Score Area(DSP) Area(LUTs) Delay Post Alloc Post Assign
  --------------------------------- ---------- --------- ---------- ----- ---------- -----------
  [Lib: ccs_ioport]
  ccs_in(3,704)                          0.000     0.000      0.000 0.000          1           1
  ccs_out(4,704)                         0.000     0.000      0.000 0.000          1           1
  ccs_sync_in_vld(6)                     0.000     0.000      0.000 0.000          1           1
  [Lib: libraries]
  Catapult.peakfinder.v1()              47.059     0.000     47.059 1.765         32          32
  [Lib: mgc_Altera-Arria-10-2_beh]
  mgc_and(1,2)                           1.000     0.000      1.000 0.260          0           4
  mgc_and(1,3)                           1.000     0.000      1.000 0.260          0           1
  mgc_nor(1,2)                           1.000     0.000      1.000 0.260          0           2
  mgc_not(1)                             0.000     0.000      0.000 0.000          0           6
  mgc_or(1,3)                            1.000     0.000      1.000 0.260          0           1
  mgc_reg_pos(1,0,0,1,1,0,0)             0.000     0.000      0.000 0.330          0           2
  mgc_reg_pos(1,0,0,1,1,1,1)             0.000     0.000      0.000 0.330          0           2
  mgc_reg_pos(22,0,0,0,0,1,1)            0.000     0.000      0.000 0.330          0          32
  mgc_reg_pos(22,0,0,1,1,1,1)            0.000     0.000      0.000 0.330          0          64
  [Lib: mgc_ioport]
  mgc_io_sync(0)                         0.000     0.000      0.000 0.000          2           2

  TOTAL AREA (After Assignment):      1513.889     0.000   1514.000
```

(a) peakfinder_top 関数のリソース推定使用量の内訳



(b) peakfinder_top 関数の回路図 (一部)

```
842     mem_1_1_sva <= 22'b0000000000000000000000;
843     mem_1_0_sva <= 22'b0000000000000000000000;
844     mem_static_init_else_asn_itm <= 22'b0000000000000000000000;
845     mem_0_0_sva <= 22'b0000000000000000000000;
846   end
847   else if ( mem_static_init_else_and_cse ) begin
848     mem_31_1_sva <= mem_31_0_sva;
849     mem_31_0_sva <= energy_adder_rsci_idat[703:682];
850     mem_30_1_sva <= mem_30_0_sva;
851     mem_30_0_sva <= energy_adder_rsci_idat[681:660];
852     mem_29_1_sva <= mem_29_0_sva;
853     mem_29_0_sva <= energy_adder_rsci_idat[659:638];
854     mem_static_init_else_asn_56_itm <= mem_28_0_sva;
855     mem_28_0_sva <= energy_adder_rsci_idat[637:616];
856     mem_static_init_else_asn_54_itm <= mem_27_0_sva;
```

(c) 生成された peakfinder_top 関数の Verilog のコード (一部)

図 5.10: RTL 生成後の peakfinder_top 関数のリソース推定使用量、回路図、Verilog のコード

5.5 RTL シミュレーション

C++シミュレーションでは 10 BC 分のランダム入力を用いた場合と 69 BC 分の主要な 729 通りの入力を用いた場合で、Peak Finder のロジックが正しいかどうか確認し、どちらも問題がなかったことを確認した。ここでは、RTL 生成後の Peak Finder について、よりハードウェア実装後のイメージに近い RTL シミュレーションの結果について述べる。

Catapult は RTL 生成後の VHDL や Verilog ファイルに対して、クロック駆動型のより実用に近い RTL

シミュレーションを実行することが可能である。そのとき使われるのが QuestaSim と呼ばれるシミュレーションツールである。QuestaSim は、Catapult が生成した HDL ファイルをシミュレーションするために、C++で記述されたオリジナルのテストベンチファイルを使う。本研究の場合は peakfinder_top_tb.cpp がこれに該当する。ただ、そのままではシミュレーションできないので、C++で記述されたテストベンチを HDL に変換する。この過程も Catapult が自動で行うので設計者の負担は減る。図 5.11 に RTL シミュレーションのテスト方法の概要図を示す。RTL シミュレーションでは、C++テストベンチからの出力と Catapult が生成した HDL テストベンチからの出力を 1 クロックごとに比較する。このとき、両者の値が一致しない場合はエラー信号を生成する。エラーがない場合はシミュレーションが成功したことを意味する。以降は VHDL によるシミュレーション結果のみを示す。

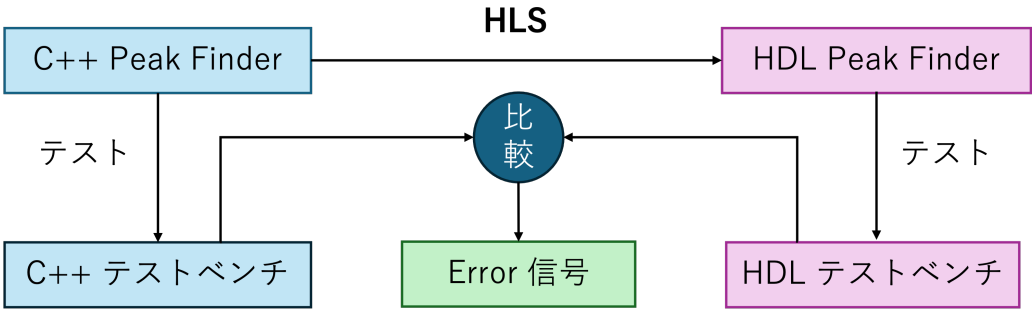


図 5.11: RTL シミュレーションのテスト方法概要図

5.5.1　ランダム入力を用いた RTL シミュレーション

図 5.12 に 10 BC 分の 0 から 10 の整数値をとるランダム入力データに対する RTL シミュレーション結果を示す。図のオレンジ色の部分は C++で記述されたオリジナルのテストベンチファイルからの出力、水色の部分は Catapult が生成した HDL ファイルからの出力を表す。上で述べたように、QuestaSim は両者の値をクロックごとに比較し、相違があればエラー信号を生成する。それが図の赤色の部分である。10 BC 全てにおいてエラーがなく、RTL シミュレーションが成功したことを確認した。

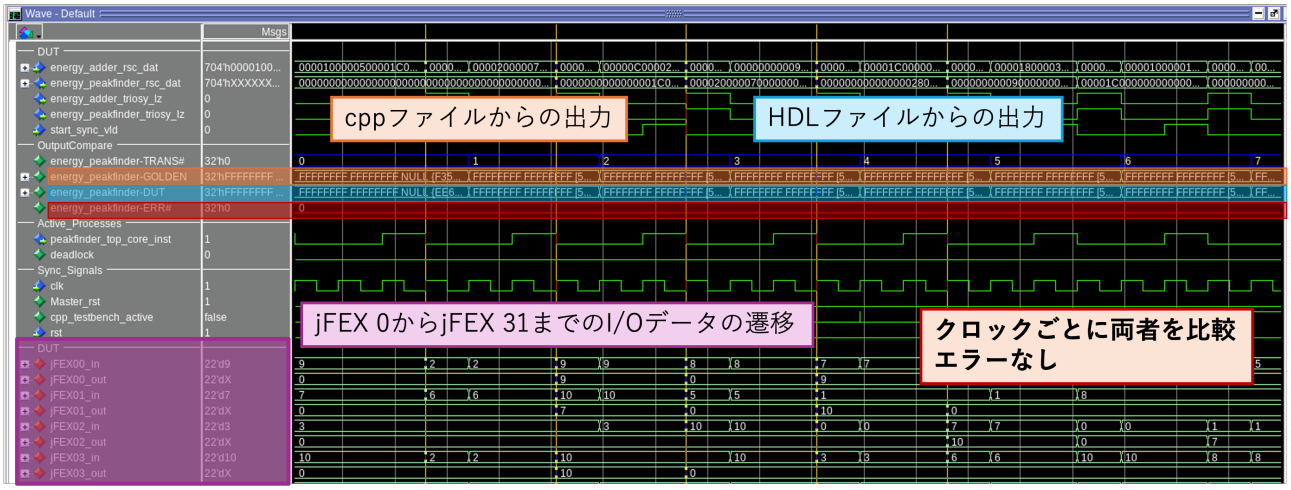


図 5.12: ランダム入力データを用いた peakfinder_top 関数の RTL シミュレーション結果

また、図中の紫色の部分は 32 個に展開された peakfinder 関数の入力と出力を表している。図 5.13 に

その部分を拡大した様子を示す。 クロックは 240 MHz に設定してある。図では、jFEX 0 と jFEX 1 についての入力と出力のデータ遷移について説明している[5]。例えば、jFEX 0 の 2, 9, 8 という並びについては中央の値 (9) はピークなので、出力は 9 になる。ただし、ピークの判定には原理的に 1 BC のレイテンシがかかるので、シミュレーション上でも出力がその分遅れているのが確認できる。jFEX 1 の例も見てみる。jFEX 1 の 10, 5, 1 という並びの中央の値 (5) はピークではないので、1 BC 遅れて 0 が出力されているのが分かる。他の jFEX についても同様だった。

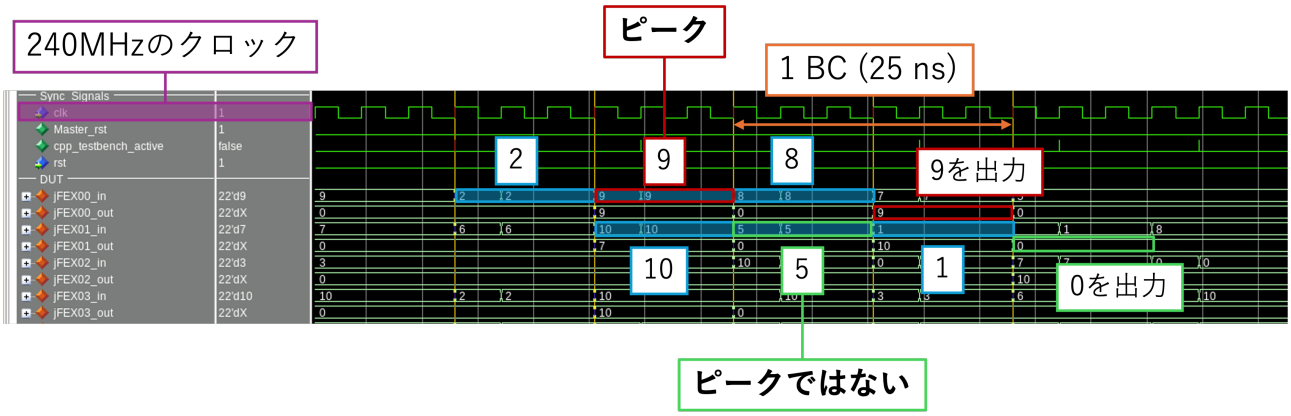


図 5.13: ランダム入力データを用いた peakfinder_top 関数の RTL シミュレーション結果 (拡大図)

上で見た 0 から 10 の入力や出力は FPGA 上では 2 進数として扱われる。そこで、データの転送が bit 単位で行われていることも確認した。図 5.14 は、jFEX 24 のある BC での出力値 (5) を bit 表記にしたものである。22 bit の bit 幅のうち、LSB から数えて 3 bit のみが 101 (2 進数の 5) に対応する信号になっているのが分かる。

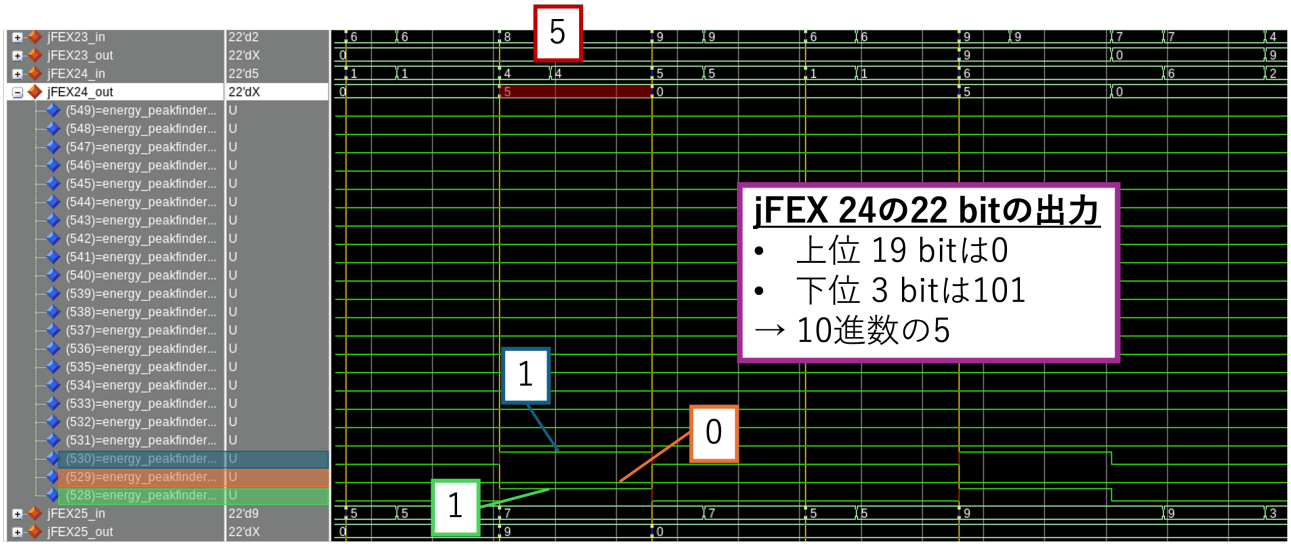


図 5.14: jFEX 24 の出力値の bit 表記

以上の結果から、10 BC 分のランダム入力データに対する RTL シミュレーションは期待通りのロジックで動作し、エラーなく成功したと結論付ける。

[5]jFEX 0 はトップレベル内で 32 個に展開された peakfinder 関数のうちループ番号 0 に対応し、jFEX 1 はループ番号 1 に対応する。

5.5.2　主要な入力組み合わせを考慮した RTL シミュレーション

ここでは、5.3.3 項で見た 729 通りの入力に対する RTL シミュレーションの結果について述べる。図 5.15 は、jFEX 0 と jFEX 2 の入力と出力のデータ遷移を示している。ここでもクロックは 240 MHz である。例えば、jFEX 0 の入力が -100,000, -1, -2,097,152 だった場合、中央の値 (-1) はピークである。したがって、1 BC のレイテンシで -1 が出力される。また、jFEX 2 の入力についても見てみる。jFEX 2 の -1, -2,097,152, -1 という並びの中央の値 (-2,097,152) はピークではないので、1 BC 遅れて 0 が出力されているのが分かる。他の jFEX についてもロジックとタイミングが正しいことを確認した。

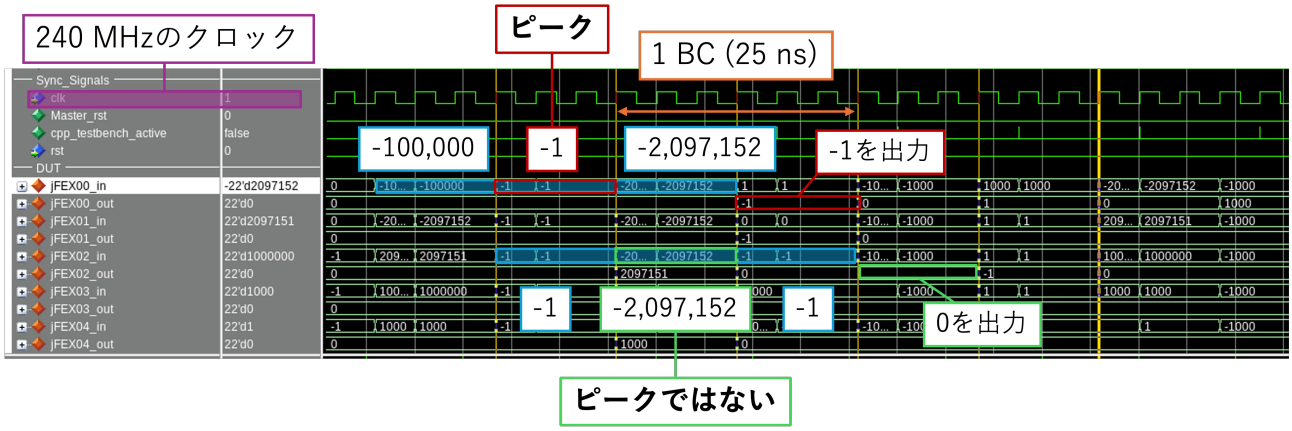


図 5.15: 主要な組み合わせを考慮した入力に対する peakfinder_top 関数の RTL シミュレーション結果

以上の結果から、69 BC 分の主要な組み合わせを考慮した入力データに対する RTL シミュレーションについても、期待通りのロジックで動作し、エラーなく成功したと結論付ける。

次章では、この peakfinder_top 関数の Osum への統合から RTL シミュレーションまで、その内容と結果について述べる。

第6章　Output Summingへの統合と検証

前章では最も簡単な Peak Finding Algorithm を実装した Peak Finder のシミュレーション結果について述べた。Peak Finder 単体の機能に問題がないことを確認したら、CERN の HLS 開発グループから受け取った Output Summing のコード[1]に Peak Finder の統合を行った。シミュレーションの流れは Peak Finder 単体の場合とほとんど同じだが、LATOME 全体での検証が可能な Layer 1 と呼ばれるシミュレーションも追加で行った。以下、各シミュレーションの内容と結果について述べる。

6.1　C++シミュレーション

HLS 開発グループから受け取った Osum 全体の処理が記述されたファイルは osum_fex.h と呼ばれるものだったが、HLS 実行時にリンキングエラーを起こしたのでファイルを分割することにした。具体的には、osum_fex 関数 (Osum の処理が記述された関数) の宣言を osum_fex.h に、osum_fex 関数の定義を osum_fex.cpp に分けた。Peak Finder の Osum への統合は peakfinder_top 関数の内容 (図 5.3) をそのまま実装する形で行った。実装場所は図 5.1 で示した通り、jFEX path の adder と jfex_mle の間である。このとき、adder からの出力 energy_adder[32] は mem[ ][0] に格納し、peakfinder からの出力 energy_peakfinder[32] は後段の jfex_mle の入力として接続した。

HLS による Peak Finder の開発でも述べたように、C++シミュレーションを行うには C++で書かれたテストベンチが必要である。そこで、osum_fex_tb.cpp という名前のテストベンチを設計した (図 6.1)。その流れを以下に示す。

[テストベンチの構造]

1. 10 BC × 320 = 3,200 個の 18 bit のランダム入力データ $E_T^{UserCode}$ を生成する[2]。
2. Osum の入出力データを初期化する。
3. Peak Finder の入出力データを保存するための CSV ファイルを準備する。
4. 10 BC 分だけ osum_fex 関数を呼び出し、Peak Finder の入出力データを CSV ファイルに保存する。
5. Osum の出力を画面に表示する。
6. CSV ファイルを閉じる。
7. シミュレーションを終了する。

[1]事前にこのオリジナルのコードが問題なく動くことを確認した。
[2]Osum の入力は 13 種類あるが、このうち User Code からのエネルギーデータは 1 BC につき 320 個送られてくる。

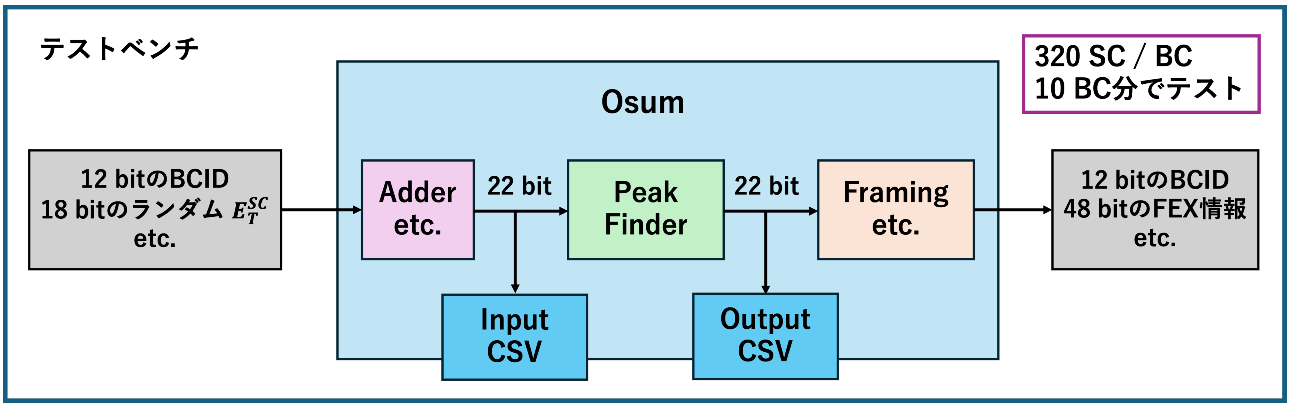


図 6.1: Peak Finder を実装した Osum のテストベンチの概要。

図 6.2 に Eclipse を用いた C++シミュレーションの結果を示す。図の左下にあるようにシミュレーションは無事に終了し、エラーは確認されなかった。また、図には非常に長い 2 進数が出力されているがこれは Osum の出力を表している。

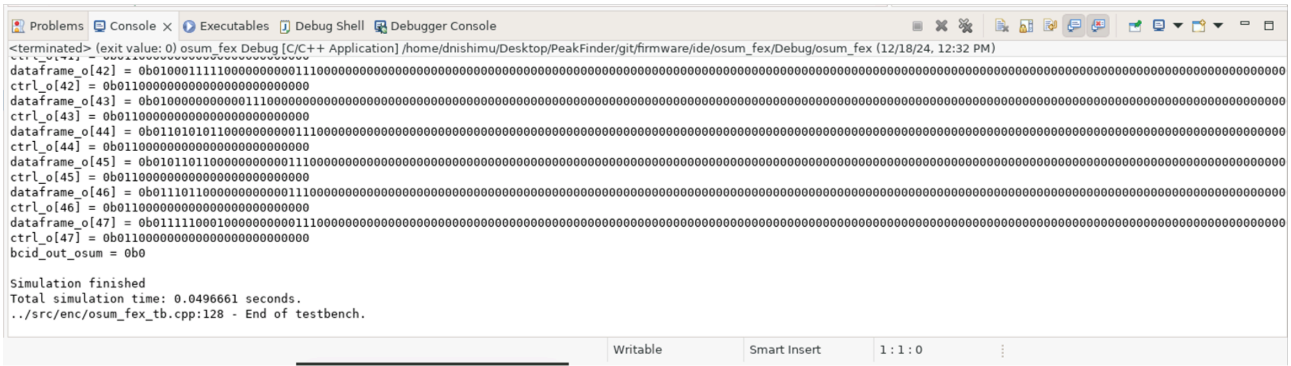


図 6.2: Peak Finder を実装した Osum の C++シミュレーション結果

また、図 6.3 に Osum に実装された peakfinder 関数の入力と出力が保存された CSV ファイルの内容を示す。これらの内容から分かるように、peakfinder 関数のロジックが正しいことも確認された。例えば、図 6.3b の jFEX 2 の BC 7 の値 (68784) を見ると、これは図 6.3a の jFEX 2 の BC 6 の値 (68784) が前後の BC と比較して最大であったため、この値が 1 BC 遅れて出力されているのが分かる。他の jFEX も同様である。

			jFEX 0,	jFEX 1,	jFEX 2,	jFEX 3,	jFEX 4,	jFEX 5,	jFEX 6,
BC	0,	43333,	396435,	-303203,	-255712,	-363442,	-430713,	-244912,	
BC	1,	125243,	-31935,	246704,	477681,	29820,	-373340,	63822,	
BC	2,	174855,	249851,	49671,	-356507,	-27785,	212388,	-46728,	
BC	3,	177532,	-46390,	217143,	-64091,	220655,	-41478,	-133759,	
BC	4,	169906,	142417,	-159188,	23407,	3204,	71878,	6722,	
BC	5,	-129163,	-11017,	-106473,	-259127,	155531,	721001,	-169937,	
BC	6,	116773,	231844,	68784,	-206148,	-296556,	293023,	194041,	
BC	7,	-169882,	-409476,	-414364,	421622,	-116490,	-114970,	313614,	
BC	8,	-214915,	276911,	-148708,	-16651,	-3596,	-185132,	225934,	
BC	9,	148647,	159182,	260387,	-32769,	95311,	-369416,	-51738,	

(a) Osum に実装された peakfinder 関数の入力

			jFEX 0,	jFEX 1,	jFEX 2,	jFEX 3,	jFEX 4,	jFEX 5,	jFEX 6,
BC	0,	0,	0,	0,	0,	0,	0,	0,	
BC	1,	0,	396435,	0,	0,	0,	0,	0,	
BC	2,	0,	0,	246704,	477681,	29820,	0,	63822,	
BC	3,	0,	249851,	0,	0,	0,	212388,	0,	
BC	4,	177532,	0,	217143,	0,	220655,	0,	0,	
BC	5,	0,	142417,	0,	23407,	0,	0,	6722,	
BC	6,	0,	0,	0,	0,	155531,	721001,	0,	
BC	7,	116773,	231844,	68784,	0,	0,	0,	0,	
BC	8,	0,	0,	0,	421622,	0,	0,	313614,	
BC	9,	0,	276911,	0,	0,	0,	0,	0,	

(b) Osum に実装された peakfinder 関数の出力

図 6.3: Osum に実装された peakfinder 関数の入力と出力

C++シミュレーションのレベルでは、Peak Finder の Osum への統合は成功した。次節では、この osum_fex 関数に対して HLS を実行し、RTL 生成した結果について述べる。

6.2 HLS による RTL 生成

6.2.1 合成の Architecture

図 6.4 に Peak Finder を実装した Osum 全体の Architecture を示す。処理の効率化のため、Osum 内の各関数の呼び出しに関する全てのループを展開し、メイン関数をパイプライン化した。トップレベルは osum_fex.cpp であり、sub block として efex_mle.cpp や adder.cpp など複数の関数を含む。また、設定した FPGA は実際に LATOME で使われているもの、すなわち Altera Arria 10 10AX115R3F40E2SG を指定し、クロックは 240 MHz に設定した。さらに、peakfinder 関数は組み合わせ回路として実装した。

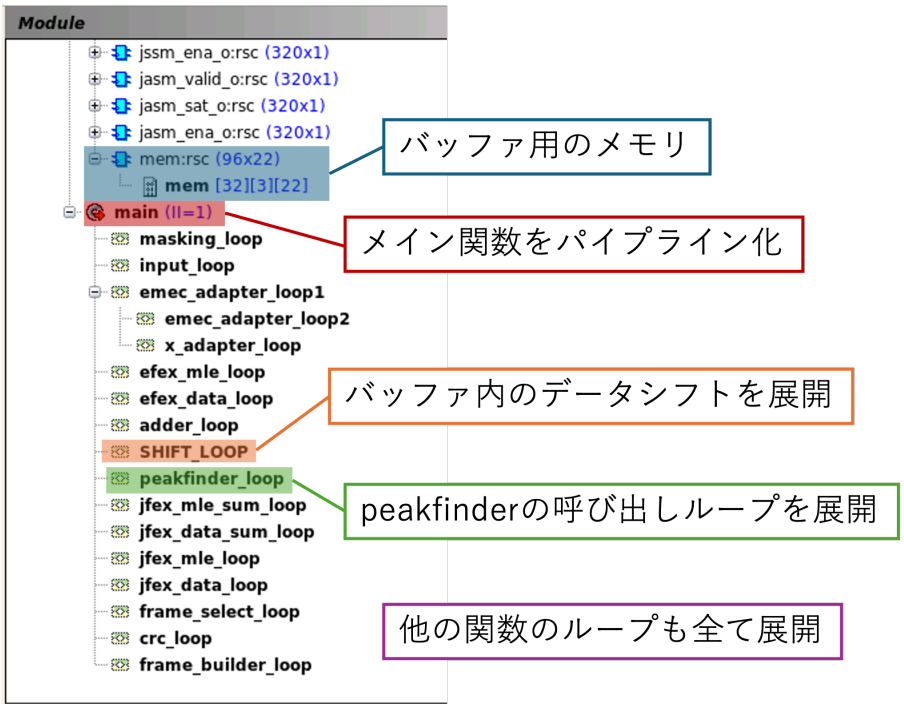


図 6.4: Peak Finder を実装した Osum の Architecture

6.2.2　RTL 生成結果

図 6.5b に Peak Finder を実装した Osum の RTL 生成結果を示す。また、比較のため図 6.5a に Peak Finder を実装する前のオリジナルの Osum の RTL 生成結果を示す。Osum のレイテンシ、スループット、Slack は Peak Finder の実装前後で変化しなかった。それぞれの値は、レイテンシが 9 クロックサイクルで 37.53 ns、スループットが 1 クロックサイクルで 4.17 ns、Slack が 0.09 ns であった。一方、リソース推定使用量は Peak Finder 実装後の Osum が 146685.88、Peak Finder 実装前の Osum が 144471.99 でその差は 2213.89 であった。この増加分が Peak Finder の実質的なリソース推定使用量に対応する。

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
efex_mle.v1 (extract)	0	0.00	0	0.00	0.42	114.53
jfex_mle.v1 (extract)	0	0.00	0	0.00	0.84	107.70
jfex_smle.v1 (extract)	0	0.00	0	0.00	0.91	101.74
emec_adapter.v1 (extract)	0	0.00	0	0.00	2.47	196.23
masking.v1 (extract)	0	0.00	0	0.00	3.65	19.00
jasm.v1 (extract)	1	4.17	1	4.17	2.85	19520.00
jssm.v1 (extract)	1	4.17	1	4.17	2.87	13120.00
adder.v1 (extract)	2	8.34	1	4.17	2.12	99.95
efex_data.v1 (extract)	0	0.00	0	0.00	4.17	0.00
jfex_data.v1 (extract)	0	0.00	0	0.00	4.17	0.00
osm.v1 (extract)	1	4.17	1	4.17	2.71	13706.44
frame_select.v1 (extract)	0	0.00	0	0.00	2.31	206.94
crc9.v1 (extract)	0	0.00	0	0.00	1.77	134.00
frame_concat.v1 (extract)	0	0.00	0	0.00	4.17	0.00
osum_fex.v1 (extract)	**9**	**37.53**	**1**	**4.17**	**0.09**	144471.99

(a) Peak Finder を含まないオリジナルの Osum の RTL 生成結果

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Slack	Total Area
efex_mle.v1 (extract)	0	0.00	0	0.00	0.42	114.53
jfex_mle.v1 (extract)	0	0.00	0	0.00	0.84	107.70
jfex_smle.v1 (extract)	0	0.00	0	0.00	0.91	101.74
emec_adapter.v1 (extract)	0	0.00	0	0.00	2.47	196.23
masking.v1 (extract)	0	0.00	0	0.00	3.65	19.00
jasm.v1 (extract)	1	4.17	1	4.17	2.85	19520.00
jssm.v1 (extract)	1	4.17	1	4.17	2.87	13120.00
adder.v1 (extract)	2	8.34	1	4.17	2.12	99.95
peakfinder.v1 (extract)	0	0.00	0	0.00	2.40	47.06
efex_data.v1 (extract)	0	0.00	0	0.00	4.17	0.00
jfex_data.v1 (extract)	0	0.00	0	0.00	4.17	0.00
osm.v1 (extract)	1	4.17	1	4.17	2.71	13706.44
frame_select.v1 (extract)	0	0.00	0	0.00	2.31	206.94
crc9.v1 (extract)	0	0.00	0	0.00	1.77	134.00
frame_concat.v1 (extract)	0	0.00	0	0.00	4.17	0.00
osum_fex.v1 (extract)	**9**	**37.53**	**1**	**4.17**	**0.09**	146685.88

(b) Peak Finder が実装された Osum の RTL 生成結果。水色で塗りつぶされた箇所が peakfinder に該当する部分である。

図 6.5: Peak Finder 実装前後の Osum の RTL 生成結果

また、Peak Finder 実装前後の Osum のリソース推定使用量の内訳をそれぞれ図 6.6 と図 6.7 に示す。これによると、Area が 47.059 の peakfinder 関数 32 個、Area が 1.000 の AND 回路 3 個、Area が 22.000 のマルチプレクサ 32 個、Area が 1.000 の OR 回路 1 個がリソースの増加に寄与しているのが分かる。この他にもレジスタの増加などが確認できるが、これらは Area が 0 である。これらの論理素子から Peak Finder の実装によって増加したリソース推定使用量を計算してみる。

$$47.059 \times 32 + 1.000 \times 3 + 22.000 \times 32 + 1.000 \times 1 = 2213.89$$

これは元の Osum のリソース推定使用量との差分に一致する。

Bill Of Materials (Datapath)

Component Name	Area Score	Area(DSP)	Area(LUTs)	Delay	Post Alloc	Post Assign
[Lib: ccs_ioport]						
ccs_in(122,5760)	0.000	0.000	0.000	0.000	1	1
ccs_in(123,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(124,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(125,16)	0.000	0.000	0.000	0.000	1	1
ccs_in(126,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(127,960)	0.000	0.000	0.000	0.000	1	1
ccs_in(128,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(129,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(130,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(131,8)	0.000	0.000	0.000	0.000	1	1
ccs_in(132,96)	0.000	0.000	0.000	0.000	1	1
ccs_in(133,32)	0.000	0.000	0.000	0.000	1	1
ccs_in(134,12)	0.000	0.000	0.000	0.000	1	1
ccs_in(135,12)	0.000	0.000	0.000	0.000	1	1
ccs_in(136,96)	0.000	0.000	0.000	0.000	1	1
ccs_out(137,10752)	0.000	0.000	0.000	0.000	1	1
ccs_out(138,1344)	0.000	0.000	0.000	0.000	1	1
ccs_out(139,12)	0.000	0.000	0.000	0.000	1	1
ccs_sync_in_vld(177)	0.000	0.000	0.000	0.000	1	1
[Lib: libraries]						
Catapult_project_2.adder.v1()	99.951	0.000	99.951	0.330	32	32
Catapult_project_2.crc9.v1()	134.000	0.000	134.000	2.400	48	48
Catapult_project_2.efex_data.v1()	0.000	0.000	0.000	0.000	16	16
Catapult_project_2.efex_mle.v1()	114.529	0.000	114.529	3.753	320	320
Catapult_project_2.emec_adapter.v1()	196.225	0.000	196.225	1.697	16	16
Catapult_project_2.frame_concat.v1()	0.000	0.000	0.000	0.000	48	48
Catapult_project_2.frame_select.v1()	206.937	0.000	206.937	1.860	48	48
Catapult_project_2.jasm.v1()	19520.000	0.000	19520.000	0.330	1	1
Catapult_project_2.jfex_data.v1()	0.000	0.000	0.000	0.000	18	18
Catapult_project_2.jfex_mle.v1()	107.702	0.000	107.702	3.331	32	32
Catapult_project_2.jfex_smle.v1()	101.738	0.000	101.738	3.262	256	256
Catapult_project_2.jssm.v1()	13120.000	0.000	13120.000	0.330	1	1
Catapult_project_2.masking.v1()	19.000	0.000	19.000	0.520	320	320
Catapult_project_2.osm.v1()	13706.436	0.000	13706.436	0.330	1	1
[Lib: mgc_Altera-Arria-10-2_beh]						
mgc_and(1,2)	1.000	0.000	1.000	0.260	0	28
mgc_and(1,3)	1.000	0.000	1.000	0.260	0	2
mgc_mux(1,1,2)	1.000	0.000	1.000	0.260	288	576
mgc_mux(18,1,2)	18.000	0.000	18.000	0.260	96	144
mgc_nor(1,2)	1.000	0.000	1.000	0.260	0	2
mgc_not(1)	0.000	0.000	0.000	0.000	0	21
mgc_or(1,3)	1.000	0.000	1.000	0.260	0	1
mgc_reg_pos(1,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	2214
mgc_reg_pos(1,0,0,1,1,0,0)	0.000	0.000	0.000	0.330	0	2
mgc_reg_pos(1,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	19
mgc_reg_pos(10,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	112
mgc_reg_pos(12,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	73
mgc_reg_pos(16,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	33
mgc_reg_pos(18,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	688
mgc_reg_pos(2,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	76
mgc_reg_pos(215,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	48
mgc_reg_pos(22,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	32
mgc_reg_pos(224,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	170
mgc_reg_pos(28,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	96
mgc_reg_pos(288,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	16
mgc_reg_pos(32,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	30
mgc_reg_pos(320,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	3
mgc_reg_pos(8,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	30
mgc_reg_pos(8,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	22
mgc_reg_pos(96,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	13
mgc_reg_pos(960,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	1
[Lib: mgc_ioport]						
mgc_io_sync(0)	0.000	0.000	0.000	0.000	18	18
TOTAL AREA (After Assignment):	144470.991	0.000	144471.000			

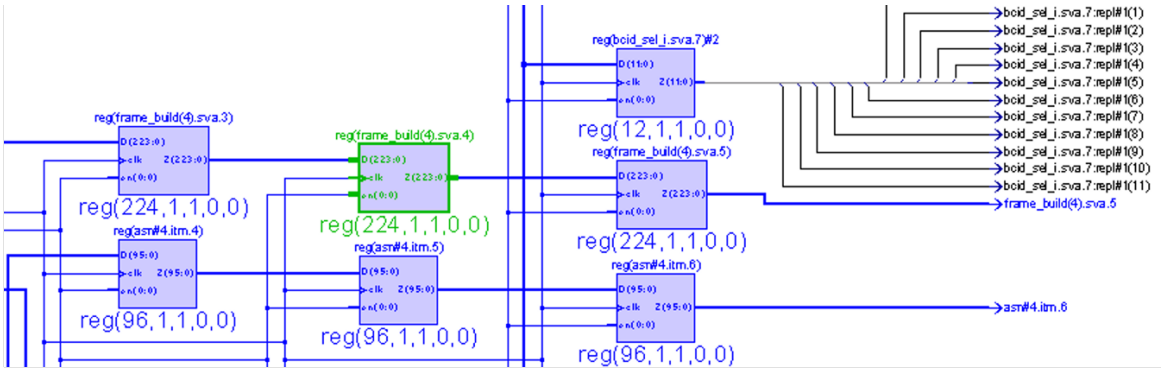
図 6.6: Peak Finder を含まないオリジナルの Osum のリソース推定使用量の内訳。Post Assign は割り当てられた論理素子の数を表す。

Bill Of Materials (Datapath)

Component Name	Area Score	Area(DSP)	Area(LUTs)	Delay	Post Alloc	Post Assign
[Lib: ccs_ioport]						
ccs_in(124,5760)	0.000	0.000	0.000	0.000	1	1
ccs_in(125,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(126,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(127,16)	0.000	0.000	0.000	0.000	1	1
ccs_in(128,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(129,960)	0.000	0.000	0.000	0.000	1	1
ccs_in(130,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(131,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(132,320)	0.000	0.000	0.000	0.000	1	1
ccs_in(133,8)	0.000	0.000	0.000	0.000	1	1
ccs_in(134,96)	0.000	0.000	0.000	0.000	1	1
ccs_in(135,32)	0.000	0.000	0.000	0.000	1	1
ccs_in(136,12)	0.000	0.000	0.000	0.000	1	1
ccs_in(137,12)	0.000	0.000	0.000	0.000	1	1
ccs_in(138,96)	0.000	0.000	0.000	0.000	1	1
ccs_out(139,10752)	0.000	0.000	0.000	0.000	1	1
ccs_out(140,1344)	0.000	0.000	0.000	0.000	1	1
ccs_out(141,12)	0.000	0.000	0.000	0.000	1	1
ccs_sync_in_vld(181)	0.000	0.000	0.000	0.000	1	1
[Lib: libraries]						
Catapult_project_17.adder.v1()	99.951	0.000	99.951	0.330	32	32
Catapult_project_17.crc9.v1()	134.000	0.000	134.000	2.400	48	48
Catapult_project_17.efex_data.v1()	0.000	0.000	0.000	0.000	16	16
Catapult_project_17.efex_mle.v1()	114.529	0.000	114.529	3.753	320	320
Catapult_project_17.emec_adapter.v1()	196.225	0.000	196.225	1.697	16	16
Catapult_project_17.frame_concat.v1()	0.000	0.000	0.000	0.000	48	48
Catapult_project_17.frame_select.v1()	206.937	0.000	206.937	1.860	48	48
Catapult_project_17.jasm.v1()	19520.000	0.000	19520.000	0.330	1	1
Catapult_project_17.jfex_data.v1()	0.000	0.000	0.000	0.000	18	18
Catapult_project_17.jfex_mle.v1()	107.702	0.000	107.702	3.331	32	32
Catapult_project_17.jfex_smle.v1()	101.738	0.000	101.738	3.262	256	256
Catapult_project_17.jssm.v1()	13120.000	0.000	13120.000	0.330	1	1
Catapult_project_17.masking.v1()	19.000	0.000	19.000	0.520	320	320
Catapult_project_17.osm.v1()	13706.436	0.000	13706.436	0.330	1	1
Catapult_project_17.peakfinder.v1()	47.059	0.000	47.059	1.765	32	32
[Lib: mgc_Altera-Arria-10-2_beh]						
mgc_and(1,2)	1.000	0.000	1.000	0.260	0	31
mgc_and(1,3)	1.000	0.000	1.000	0.260	0	2
mgc_mux(1,1,2)	1.000	0.000	1.000	0.260	288	576
mgc_mux(18,1,2)	18.000	0.000	18.000	0.260	96	144
mgc_mux(22,1,2)	22.000	0.000	22.000	0.260	0	32
mgc_nor(1,2)	1.000	0.000	1.000	0.260	0	2
mgc_not(1)	0.000	0.000	0.000	0.000	0	22
mgc_or(1,2)	1.000	0.000	1.000	0.260	0	1
mgc_or(1,3)	1.000	0.000	1.000	0.260	0	1
mgc_reg_pos(1,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	2214
mgc_reg_pos(1,0,0,1,1,0,0)	0.000	0.000	0.000	0.330	0	2
mgc_reg_pos(1,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	20
mgc_reg_pos(10,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	112
mgc_reg_pos(12,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	73
mgc_reg_pos(16,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	33
mgc_reg_pos(18,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	688
mgc_reg_pos(2,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	76
mgc_reg_pos(215,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	48
mgc_reg_pos(22,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	64
mgc_reg_pos(22,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	64
mgc_reg_pos(224,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	170
mgc_reg_pos(28,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	96
mgc_reg_pos(288,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	16
mgc_reg_pos(32,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	30
mgc_reg_pos(320,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	3
mgc_reg_pos(8,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	30
mgc_reg_pos(8,0,0,1,1,1,1)	0.000	0.000	0.000	0.330	0	22
mgc_reg_pos(96,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	13
mgc_reg_pos(960,0,0,0,0,1,1)	0.000	0.000	0.000	0.330	0	1
[Lib: mgc_ioport]						
mgc_io_sync(0)	0.000	0.000	0.000	0.000	18	18
TOTAL AREA (After Assignment):	146684.880	0.000	146685.000			

図 6.7: Peak Finder が実装された Osum のリソース推定使用量の内訳。図中の赤色で塗りつぶされた箇所が新たに増えたリソースを表している。AND 回路は 3 つ数が増えたのが確認できる。

最後に、RTL と共に生成された Osum の回路図と Verilog のコードを図 6.8 に示す。Osum 全体の回路図はとても複雑であり、ここではその詳細については省略する。



(a) Peak Finder が実装された Osum の回路図 (一部)。いくつかのレジスタが確認できる。

```
9795  wire [65:0] nl_peakfinder_loop_20_peakfinder_rg_sum_ET;
9796  assign nl_peakfinder_loop_20_peakfinder_rg_sum_ET = {mem_static_init_else_asn_38_itm_1
9797      , mem_19_0_sva , adder_loop_1_adder_cmp_13_data_o_rsc_z};
9798  wire [65:0] nl_peakfinder_loop_19_peakfinder_rg_sum_ET;
9799  assign nl_peakfinder_loop_19_peakfinder_rg_sum_ET = {mem_static_init_else_asn_36_itm_1
9800      , mem_18_0_sva , adder_loop_1_adder_cmp_14_data_o_rsc_z};
9801  wire [65:0] nl_peakfinder_loop_18_peakfinder_rg_sum_ET;
9802  assign nl_peakfinder_loop_18_peakfinder_rg_sum_ET = {mem_static_init_else_asn_34_itm_1
9803      , mem_17_0_sva , adder_loop_1_adder_cmp_15_data_o_rsc_z};
9804  wire [65:0] nl_peakfinder_loop_17_peakfinder_rg_sum_ET;
9805  assign nl_peakfinder_loop_17_peakfinder_rg_sum_ET = {mem_static_init_else_asn_32_itm_1
9806      , mem_16_0_sva , adder_loop_1_adder_cmp_16_data_o_rsc_z};
9807  wire [65:0] nl_peakfinder_loop_16_peakfinder_rg_sum_ET;
9808  assign nl_peakfinder_loop_16_peakfinder_rg_sum_ET = {mem_static_init_else_asn_30_itm_1
9809      , mem_15_0_sva , adder_loop_1_adder_cmp_17_data_o_rsc_z};
```

(b) Peak Finder が実装された Osum に対応する生成された Verilog コード (一部)。各 peakfinder 関数のレジスタに 3 つのデータを渡す部分を表している。コードは全部で 3 万行を超える。

図 6.8: Peak Finder 実装後の Osum の回路図と生成された Verilog コード

6.3　RTL シミュレーション

ここまで、Peak Finder が実装された Osum の C++シミュレーションと HLS による RTL 生成が成功したことを見た。そこで次のステップとして、Osum の RTL シミュレーションを行った。Osum への入力は C++シミュレーションのときと同様に、10 BC × 320 = 3,200 個の 18 bit のランダムデータを User Code からの入力データとし、テストベンチ内で osum_fex 関数を 10 回呼び出した。図 6.9 にその RTL シミュレーション結果を示す。図のオレンジ色の枠の中が osum_fex 関数の入出力データの遷移を、その下の赤色の枠の中が osum_fex 関数の出力値のエラー情報を示している。エラーの確認は、Peak Finder の開発と同様で、C++のテストベンチファイルからの出力と HLS 実行後に生成された HDL ファイル (この図では VHDL) を元にクロックを駆動させながら得た出力を、クロックごとに比較するという方法をとっている。このシミュレーションの結果から、Peak Finder 実装後の Osum が RTL シミュレーションをエラーなくパスしたことが確認できた。

また、Osum 内の Peak Finder のデータ遷移の様子を図 6.10 に示す[3]。図中では、jFEX 0 について説

[3]図では、osum_fex の呼び出しに 11 クロックサイクル (> 1 BC) かかっているが、このシミュレーションは “Untimed” と呼ばれるタイプであり、関数を呼び出すタイミングを意図的に設定できない。“Untimed” ではないシミュレーションは後で

明を加えた。設計では、osum_fex が呼び出されるごとにバッファ内でデータのシフティングが行われ、adder からの入力は最も新しい BC に対応する mem0 に格納される。これらの操作は並列に実行される。また、Peak Finder は組み合わせ回路として実装したので、ピークの判断はレイテンシ 0 で行われる。図中では、その様子を赤色の枠で示している。その後、jfex_mle にデータが転送される。他の jFEX についてもロジックが正しいことが確認できた。

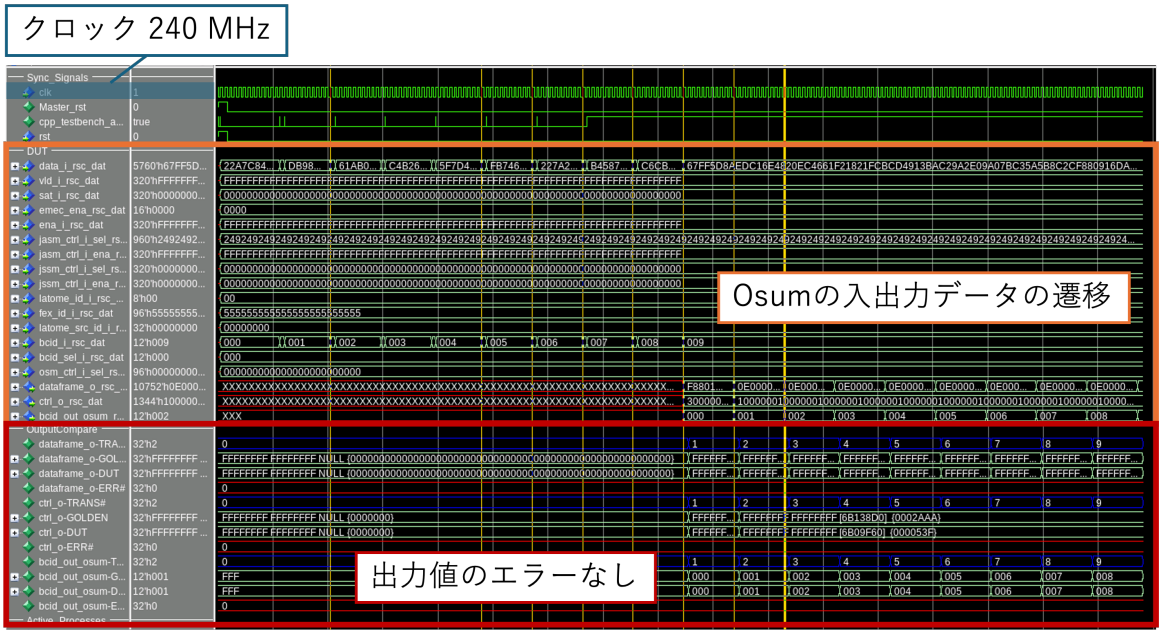


図 6.9: Peak Finder が実装された Osum の RTL シミュレーション結果

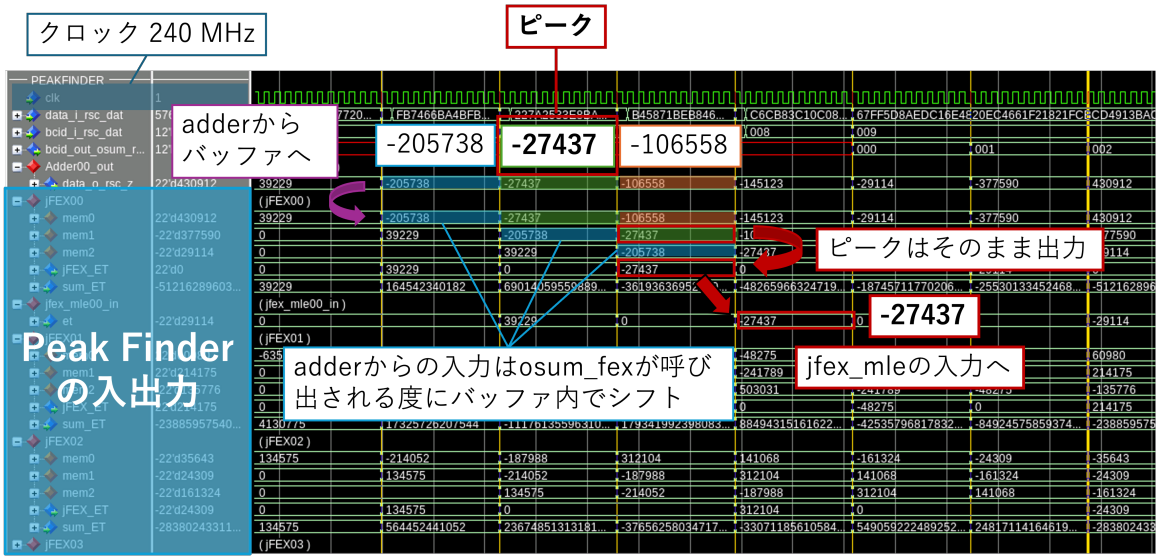


図 6.10: Osum 内の Peak Finder のデータ遷移

述べる Layer 1 である。

6.4　Layer 1 シミュレーション

Osum レベルの検証後は、LATOME 全体でのシミュレーションに進んだ。LATOME 全体でのシミュレーションには、Layer 1 から Layer 3 までの 3 つのステップがある。Layer 3 が最も厳しい検証であり、LATOME 内の全ての block を含む、完全なリアルタイムデータパスのテストが可能である。一方、Layer 1 と Layer 2 は LATOME 内で HLS 化された block のみでテストを行い、HLS 化されていない block についてはデータをバイパスする仕様になっている。ここでは、Layer 1 シミュレーションの結果について述べる。

6.4.1　シミュレーションの流れ

まず、準備として全てのシミュレーション過程を自動化する CI (Continuous Integration) に本研究で開発した peakfinder block を追加した。こうすることで、C++シミュレーション、HLS、RTL シミュレーションが一括して実行可能になる。Layer 1 シミュレーションはこの CI を利用する。図 6.11 に Layer 1 シミュレーションの対象となる block の概要図を示す。左側の Input Generator は LATOME の入力を生成する。その後、Remap に入り、Input Switch Matrix (ISM) と呼ばれる block でデータの並べ替えが行われる。この部分はすでに HLS 化されている。並べ替えの前後でデータの数は 384 から 320 に減少する。そして、User Code では何もしないまま通過し、Peak Finder が実装された Osum にデータが送られ、最終的に FEX へ向かう出力が得られる。

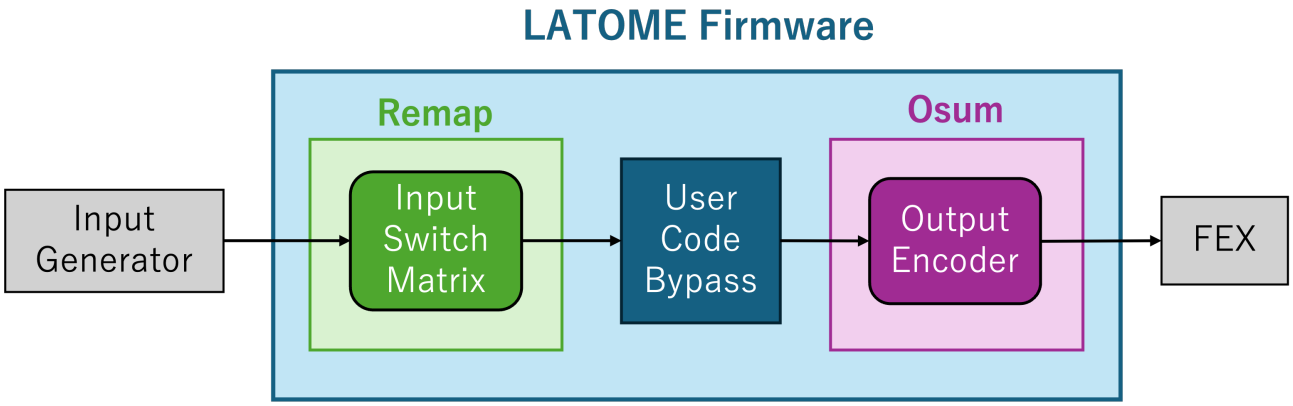


図 6.11: Layer 1 シミュレーションの概要図

Layer 1 シミュレーションでは、信号の流れを検証する方法として、以下に述べる 2 つの経路からの出力を比較する。1 つ目は、ISM と Osum について HLS によって生成された HDL ファイルを使ったシミュレーションで得られた最終的な LATOME の出力である。2 つ目は、Python で記述された LATOME からの出力である。しかし、1 つ留意すべき点がある。それは、この Python コードには Peak Finder のモデルが存在しないことである[4]。このことは、HDL ファイルからの出力と Python コードからの出力に違いを生むため、必然的にシミュレーションは失敗する。しかしながら、LATOME 上で Peak Finder

[4]時間の都合上、Python コードに Peak Finder のモデルを実装することができなかった。しかし、将来的には実装予定である。

の入力と出力のデータ遷移の様子は確認できるため、そのロジックが正しいかどうか検証を行った。次項ではその検証結果について述べる。

6.4.2　Layer 1 での RTL シミュレーション

図 6.12 に Layer 1 での RTL シミュレーション結果を示す。この図にはシミュレーションの一部しか映っていないが、クロック、BCID、Osum の入出力データの遷移などが確認できる。BCID の間隔は 6 クロックサイクルで 25 ns である。また、このシミュレーションでは 12 bit のランダムな ADC 値を LATOME への入力としている。シミュレーションは失敗しているが、Peak Finder の信号の遷移を確認した。図 6.13 にその様子を示す。その内容は図 6.10 と同様で、jFEX 0 についてのデータの遷移を表している。左側の mem0, mem1, mem2 は peakfinder 関数の入力として渡され、jFEX_ET は peakfinder 関数の出力である。これを見る限り、少なくとも Peak Finder のロジックが正しいことが分かる。Peak Finder で検出されたピークは 1 BC 遅れて後段の jfex_mle に送られる。他の jFEX についてもそのロジックが正しいことを確認した。

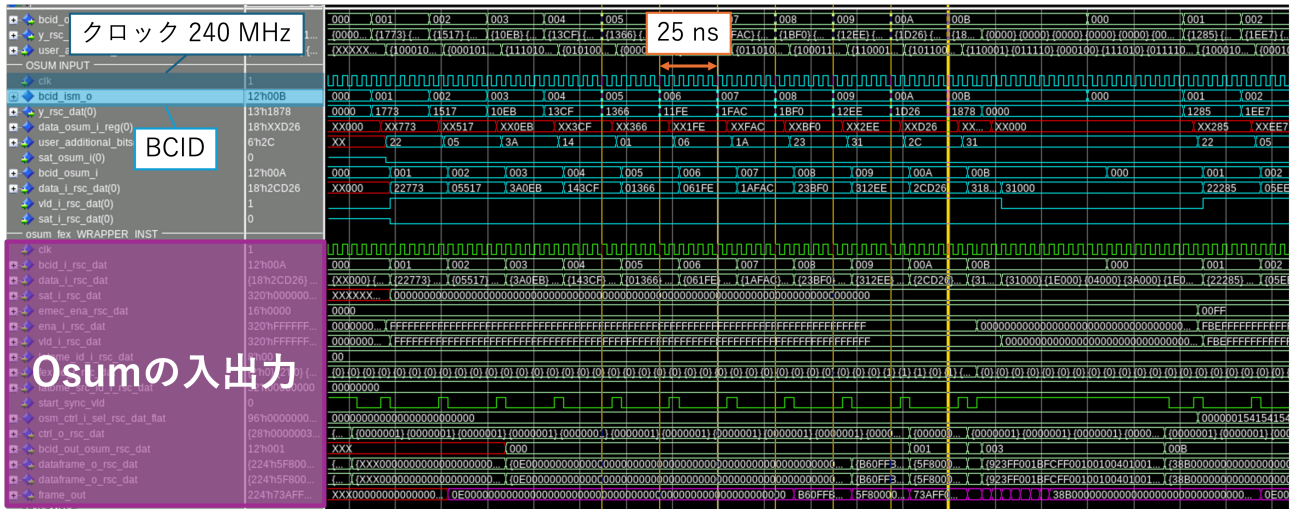


図 6.12: Layer 1 の RTL シミュレーション結果

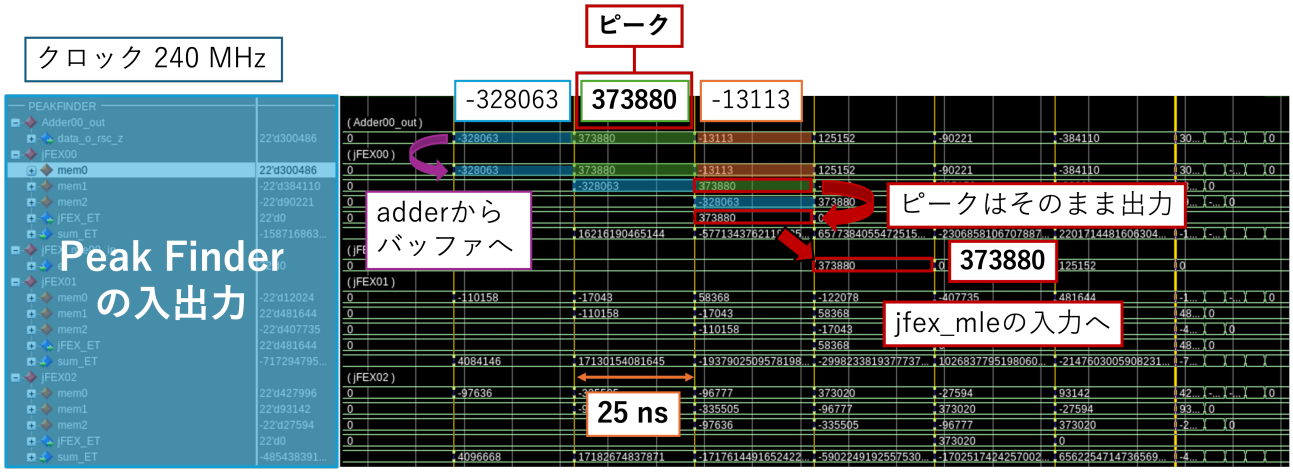


図 6.13: Layer 1 の RTL シミュレーション結果 (Peak Finder の信号)

以上の結果をまとめると、式 (5.1) で表される最もシンプルな Peak Finding Algorithm を実装した Peak Finder を Osum に統合し、RTL シミュレーションに成功することを確認した (図 6.10)。また、統合の際、リソース推定使用量がオリジナルの Osum と比較して 2213.89 増加することも確認した (全体の約 1.5%)。さらに、Layer 1 のレベルで Peak Finder が実装された Osum を LATOME に統合することに成功した。一方、Python コードに Peak Finder のモデルが存在しなかったため RTL シミュレーションは失敗に終わった。しかし、QuestaSim の Window 上で Peak Finder のロジックが期待通りであることは確認できた (図 6.13)。

第7章　結論と今後の展望

この章では、本研究で得られた結論と今後の展望について述べる。

7.1　結論

LHC の重イオン衝突実験では、UPC 由来の低エネルギーの電子や光子に対して、液体アルゴンカロリメータの現行のデジタルトリガーシステムでトリガー効率が低いことが報告されている。そこでこの問題に対処するため、重イオン衝突向けの新しいトリガー発行アルゴリズム「Peak Finding Algorithm」を LATOME ファームウェアに実装することを目的として本研究に取り組んだ。具体的には、実データを用いたアルゴリズムの妥当性評価から HLS による Peak Finder の開発、Osum への統合、そして LATOME への統合と検証を行った。これらについて、本研究で得られた結論を以下にまとめる。

重イオン衝突実験データを用いた Peak Finding Algorithm の妥当性評価

本研究では Peak Finder の開発をする前に、2023 年の秋に取得された重イオン衝突データを用いて Peak Finding Algorithm の検証を行った。Peak Finding Algorithm は、陽子・陽子衝突で使われている Super Cell 単位のトリガー読み出しではなく、$\Delta\eta \times \Delta\phi = 0.1 \times 0.1$ の jFEX と呼ばれる領域で Super Cell の信号を足し合わせ、前後の BC と比較してエネルギーが最大であるときの BC を採用し、その時のエネルギーを後段に送信するというアルゴリズムである。実データを用いた検証では、UPC の物理で重要になる 2 GeV から 5 GeV のエネルギー領域で、92.2% (統計誤差は ±0.774%) の jFEX が BCID を正しく同定できることを確認した。また、jFEX のメイン読み出し横エネルギー $main\ E_T^{jFEX}$ が 1.2 GeV 以上であれば、BCID が正しく同定された jFEX のうち 90% 以上の jFEX が $main\ E_T^{jFEX}$ とのずれが 50% 以内に収まる範囲で jFEX のエネルギーを再構成できることを確認した。

HLS による Peak Finder の開発

最もシンプルな Peak Finding Algorithm を実装した Peak Finder を C++で開発した。その後、HLS による RTL 生成と RTL シミュレーションに成功した。シミュレーションは 0 から 10 までのランダムな整数入力と 729 通りの主要な組み合わせを持つ入力の 2 つのパターンで行い、データ遷移のタイミングとロジックが正しいことを確認した。

Peak Finder の Osum への統合

HLS 開発グループから受け取った Osum の C++コードの adder と jfex_mle の間に Peak Finder を実装した。その後、C++でテストベンチを設計し、HLS による RTL 生成と RTL シミュレーションに成功した。また、Peak Finder の実装によって増えたリソース推定使用量が 2213.89 (全体の約 1.5%) であることを確認した。

Peak Finder が実装された Osum の LATOME への統合 (Layer 1)

LATOME への統合の最初のステップとして Layer 1 と呼ばれるシミュレーションを行った。統合対象の LATOME は Remap と Osum のみが HLS で実装されており、User Code はバイパスする仕様になっている。Layer 1 では Peak Finder を CI に加えることでシミュレーションの過程を自動化し、RTL シミュレーションを行った。Peak Finder のモデルが Python で実装されていなかったのでシミュレーションは失敗したが、Peak Finder 自体のロジックとデータ転送のタイミングに問題がないことは確認できた。

7.2 今後の展望

HLS による Peak Finder の開発及び LATOME への実装にあたっては多くの課題が残っている。

UPC 由来の電子・光子は液体アルゴンカロリメータに 2 GeV から 5 GeV の低いエネルギーを落とす。エネルギーが低ければ低いほどノイズなどの影響を受けやすい。そのため、例えば、宇宙線由来のノイズが Peak Finder の出力に与える揺らぎの程度について、追加の検証が必要である。また、より深い理解のためにもミューオントリガーだけでなく電子・光子のトリガーがかかったイベントについても第 4 章で述べたような検証を行う必要がある。

本研究で開発した Peak Finder は最もシンプルなアルゴリズムを採用していた。しかし、実際のデータはピークが連続する場合やノイズを含む低いエネルギーが 1 BC 空けて 2 つのピークを持つような場合も想定される。これら現実的な問題に対処するためにはアルゴリズムの改善が必要である。また、第 5 章で開発した Peak Finder はデータの入出力のみだったが、ピークを検出できたかどうかを知らせる 1 bit の enable 信号も追加予定である。そして、最も重要な課題は、Peak Finder によって jFEX で同定された BCID の情報をその jFEX を構成する各 Super Cell や eFEX/gFEX にどのように伝搬させるかである。一般に、FPGA のリソースとレイテンシはトレードオフの関係にあるため、この課題を解決するためには HLS の利点を最大限活用しながら最適な Architecture の設計を行わなければならない。最終的には、Layer 1 から Layer 3 まで順にシミュレーションを行い、LATOME への完全な実装を目指す。

謝辞

まずは主査の田中純一教授、副査の奥村恭幸准教授、末原大幹特任准教授にはお忙しい中本論文の審査をお引き受けいただき感謝申し上げます。

本研究を進めるにあたり、多大なご指導と貴重な助言を賜りました指導教員の田中純一教授に深く感謝申し上げます。特に学会発表や CERN の液体アルゴン会合の発表において、発表資料の添削と丁寧な助言をいただきました。そのおかげで、自信を持って発表に臨むことができました。同研究室の楊易霖特任助教には、ファームウェア開発のコンパイルに関する多くのアドバイスをいただきました。ここに深く感謝申し上げます。KEK 素核研の江成祐二准教授には、研究の進め方について多くのご指導を賜りました。また、研究に関する相談も快く聞いてくださり、非常に有益な助言をいただきました。この場をお借りして深くお礼申し上げます。

また、CERN 滞在中には HLS 開発グループの方々に多大なご協力を賜りました。心より感謝申し上げます。ブルックヘブン国立研究所の Marcos Vinicius Silva Oliveira には HLS を用いた開発の基礎からシミュレーションのやり方まで全面的にご指導賜りました。Huacheng Cai と Nick Fritzche とはファームウェア開発に関する非常に有意義な議論をさせていただきました。特に同室の Nick Fritzche には、幾度となく質問に対応していただいた上、Eclipse の使い方やテストケースのアイデアの提案など多大なお力添えをいただきました。また、Lucca Viccini, Melissa Aguiar, Dabson Ferreira Dos Santos には研究に関するアドバイスはもちろん、ランチにお誘いいただいたり、ポルトガル語やブラジルの文化について教えていただいたりと、生活面でも多大なご支援をいただきました。そのおかげで、楽しみながら研究を進めることができました。

同研究室の先輩にあたる張庭宇さん、藏嘉琦さん、古川真林さん、呉鍵さんには大変お世話になりました。特に古川さんには、お忙しい中数多くの質問に丁寧にご対応いただき、研究に対する不安を感じていた私を温かく励ましてくださいました。心より感謝申し上げます。ICEPP のメンバーと出張の手続きをしていただいた秘書室の方々にも深く感謝申し上げます。先輩である奥村研究室の成川佳史さんには、進路相談に乗っていただいたり、CERN 滞在中には紅葉を見に連れて行っていただいたりと、大変お世話になりました。また、苦難を共にした同期にも深く感謝申し上げます。特に、近藤翔太さん、前野伶太さん、牧田藍瑠さんには、研究の進捗を気にかけていただいたり、時には一緒に悩みを共有したりすることで、大きな励みとなりました。また、一緒に食事をしたり、リフレッシュの時間を過ごしたりと、研究生活をより充実したものにしていただきました。

最後に、悩んでいた私を心から支えてくれた家族、親戚、友人に深く感謝申し上げます。皆様の期待があったからこそ、どんなに困難な場面でも前に進む力を得ることができました。優しく見守ってくださったこと、心より感謝しております。ありがとうございました。

参考文献

[1] CERN The ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *arXiv preprint arXiv:1207.7214*, 2012. https://www.sciencedirect.com/science/article/pii/S037026931200857X?via%3Dihub.

[2] Cush. Standard Model of Elementary Particles, 2024. https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg#filelinks.

[3] Klaudia Maj. BSM physics using photon-photon fusion processes in UPC in Pb+Pb collisions with the ATLAS detector, 2023. https://arxiv.org/abs/2307.07481.

[4] CERN Service graphique. Overall view of the LHC. 2014. https://cds.cern.ch/record/1708849?ln=ja.

[5] LHC / HL-LHC Plan, 2024. https://hilumilhc.web.cern.ch/content/hl-lhc-project.

[6] HeatherGray. Public ATLAS Luminosity Results for Run 1-3 of the LHC, 2024. https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResults.

[7] ATLAS Collaboration. The ATLAS Trigger System for LHC Run 3 and Trigger performance in 2022, 2024. https://arxiv.org/pdf/2401.06630.

[8] ATLAS Collaboration. ATLAS Liquid Argon Calorimeter Phase-I Upgrade: Technical Design Report. Atlas-tdr-022; cern-lhcc-2013-017, CERN, 2013. https://cds.cern.ch/record/1602230?ln=ja.

[9] Joao Pequenao. Computer generated image of the whole ATLAS detector, 2008. https://cds.cern.ch/record/1095924?ln=ja#.

[10] Sascha Mehlhase. ATLAS detector slice (and particle visualisations), 2021. https://cds.cern.ch/record/2770815.

[11] The ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation*, Vol. 3, No. 08, p. S08003, 2008. https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08003.

[12] 大石玲誉. LHC-ATLAS RUN3 実験に向けた液体アルゴンカロリメータートリガー読み出しファームウェアの開発と検証. Master's thesis, 東京大学大学院理学系研究科, 2020.

[13] Joao Pequenao. Computer generated image of the ATLAS Liquid Argon. https://cds.cern.ch/record/1095928, 2008.

[14] ATLAS electromagnetic calorimeter layer. https://cds.cern.ch/record/39737, 1993.

[15] Atlas Collaboration, et al. Drift Time Measurement in the ATLAS Liquid Argon Electromagnetic Calorimeter using Cosmic Muons. *arXiv preprint arXiv:1002.4189*, 2010. https://arxiv.org/abs/1002.4189.

[16] 舘野元. LHC-ATLAS 実験 Phase-I アップグレードにおける液体アルゴンカロリメータトリガー読み出しの較正と性能評価. Master's thesis, 東京大学大学院理学系研究科, 2020.

[17] 仲間聖. LHC-ATLAS RUN3 実験に向けた液体アルゴンカロリメーターにおける波形異常検知システムの研究開発. Master's thesis, 東京大学大学院理学系研究科, 2021.

[18] Georges Aad, AV Akimov, K Al Khoury, M Aleksa, T Andeen, C Anelli, N Aranzabal, C Armijo, A Bagulia, J Ban, et al. The Phase-I trigger readout electronics upgrade of the ATLAS Liquid Argon calorimeters. *Journal of Instrumentation*, Vol. 17, No. 05, p. P05024, 2022. https://iopscience.iop.org/article/10.1088/1748-0221/17/05/P05024.

[19] CERN The ATLAS Colaboration. ATLAS LAr Calorimeter trigger electronics phase I upgrade: LATOME Firmware Specification, 2024. https://gitlab.cern.ch/atlas-lar-be-firmware/LATOME/LATOME-documentation/-/blob/master/LAr-LATOME-FW/LAr-LATOME-FW.pdf.

[20] The ATLAS Collabration. Prospects for Measurements of Photon-Induced Processes in Ultra-Peripheral Collisions of Heavy Ions with the ATLAS Detector in the LHC Runs 3 and 4. Technical report, CERN, Geneva, 2018. https://cds.cern.ch/record/2641655.

[21] 宇野健太. LHC-ATLAS 実験における液体アルゴンカロリメータのアップグレードに向けたファームウェアの研究開発. Master's thesis, 東京大学大学院理学系研究科, 2016.

[22] Eric Torrence. ATLAS Data Summary, 2024. https://atlasop.cern.ch/page.php?page=https://atlas-datasummary.web.cern.ch#.