HL-LHC ATLAS実験 初段ミューオントリガー論理回路開発
- 実機プロトタイプを用いた試験システムの高度化

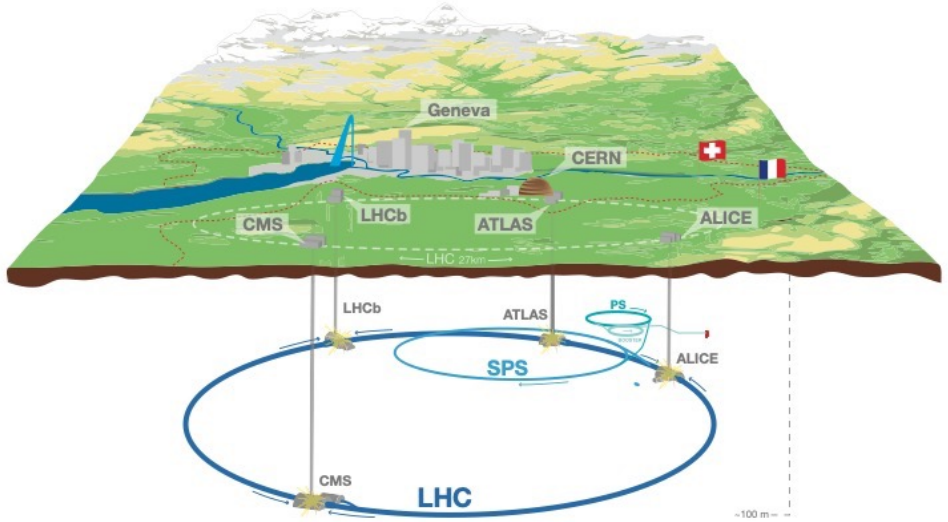
東京大学 山下恵理香

ATLAS日本トリガーグループ

LHC-ATLAS実験

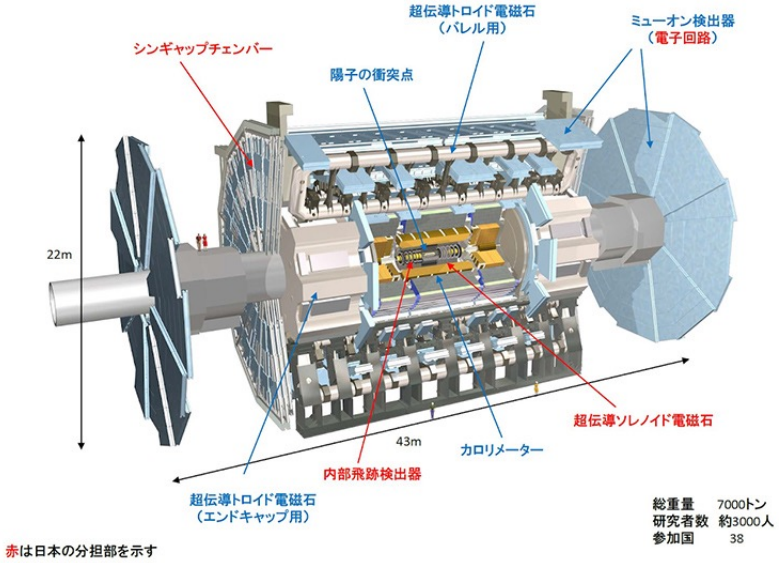
LHC加速器(Large Hadron Collider)

- 世界最高エネルギーの13.6TeV(Run3)で陽子・陽子衝突を起こし、新物理を探すエネルギーフロンティア実験
- 40 MHzで陽子を交差させる
- 2029年から高輝度LHC実験（HL-LHC）が始まる



ATLAS検出器

- LHCの陽子・陽子衝突で生成された粒子を観測するための大型汎用検出器
  - 複数の種類の検出器で構成される
- HL-LHC に向けた”Phase2アップグレード”が進行中
  - 本講演はPhase2アップグレードで開発中のエレクトロニクスの話

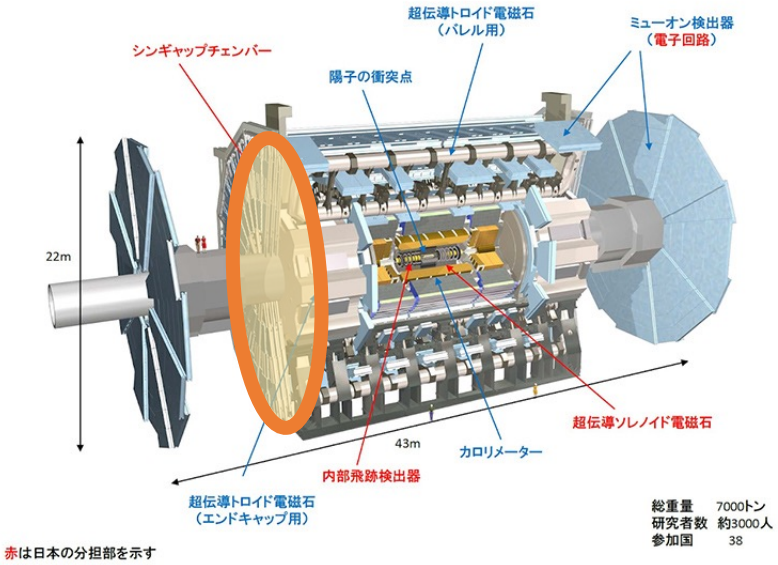


ATLAS検出機の全体図 (https://atlas.kek.jp/main/research_summary/index.html)

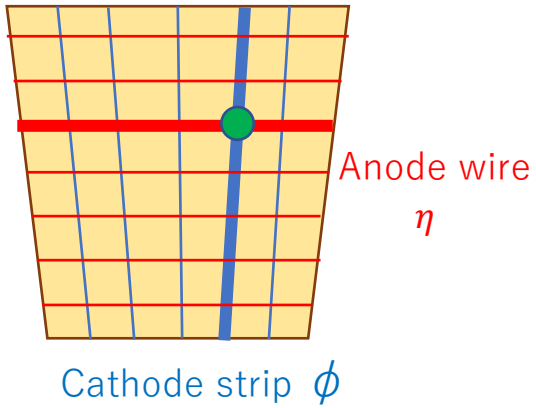
TGC検出器

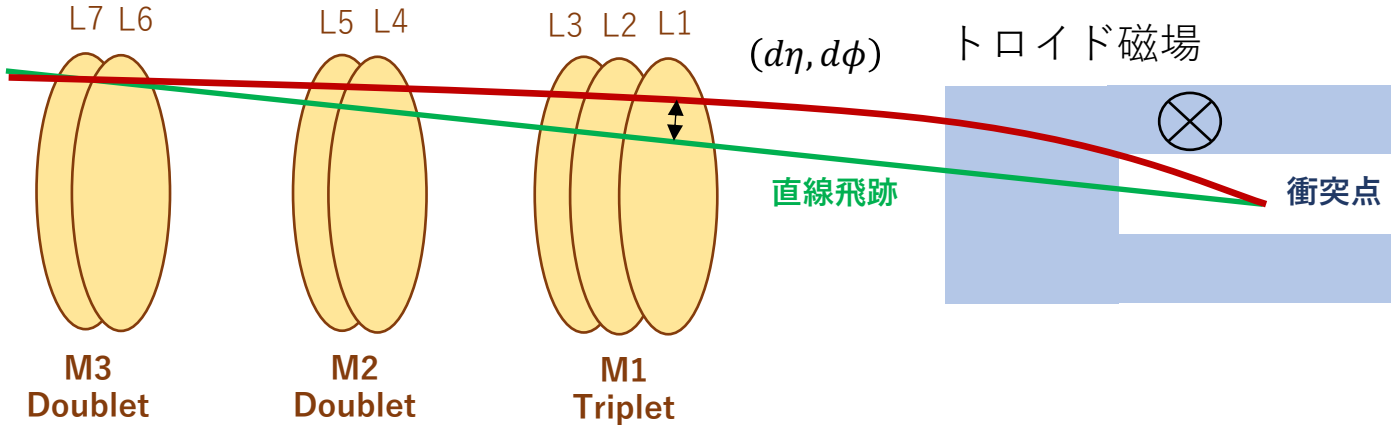
TGC(Thin Gap Chamber)とエンドキャップミューオントリガー

- 高い横運動量p_Tを持つミューオンを含む事象を選別するための トリガーとして使用
- アノードワイヤーとカソードストリップによる二次元読み出しを行う MWPC
- 7層・3ステーションで構成され、コインシデンス回路で 高速飛跡再構成（直線飛跡再構成）を行う
  - Wire 7層+Strip 6層
- トロイド磁場による曲率と直線飛跡との差分$(d\eta, d\phi)$の相関関係を 利用し、横運動量p_Tを計算する



ATLAS検出機の全体図
(https://atlas.kek.jp/main/research_summary/index.html)

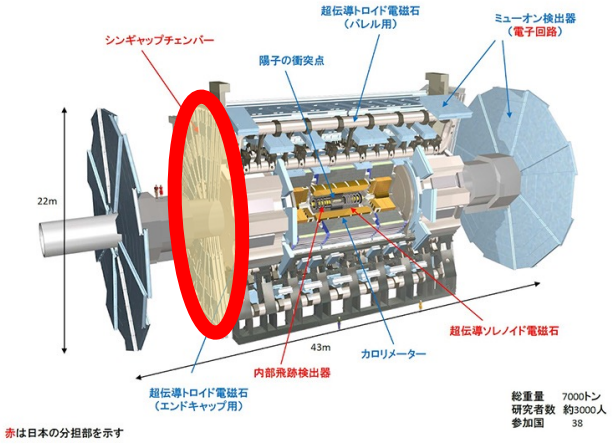




LHC-ATLAS実験のトリガーシステム

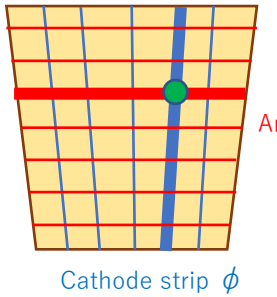
- 40 MHzの陽子交差の内、ほとんどは物理的な興味の薄い現象(陽子の非弾性散乱)
  - 記録速度・保存できるデータ量の制限から、すべてのイベントを記録することはできない
    →物理的に面白い事象のみを選別する
  - トリガーによって観測と並行して（オンラインで）事象を選別する
- ATLASのトリガーは2段階で行われるが、
  本研究の対象は初段トリガーの一部のエンドキャップミューオントリガー
  - 10 μsレイテンシーでミューオンの飛跡を再構成し、事象選別に活用する

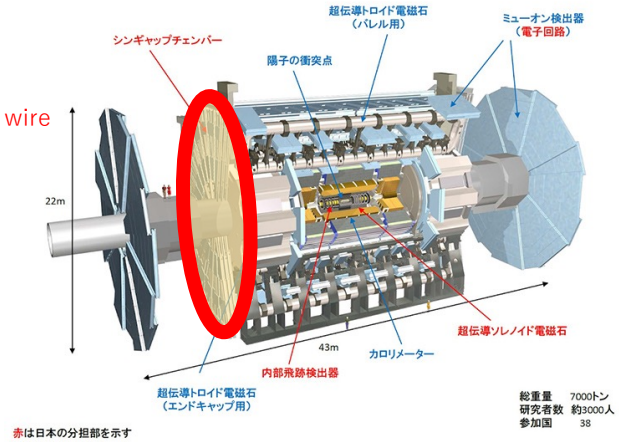
- 開発したいトリガーエレクトロニクス
  - 限られた時間内 (10 μs)で実行可能な程度にシンプルな計算かつ高効率のトリガーロジック

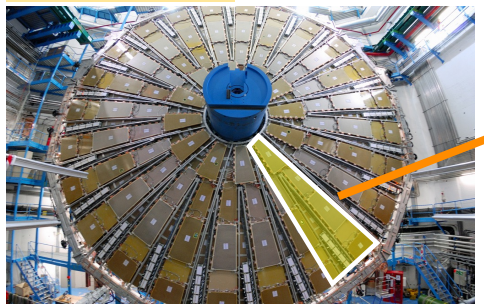


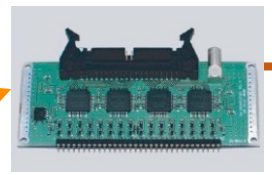
SLトリガー論理回路への入力情報

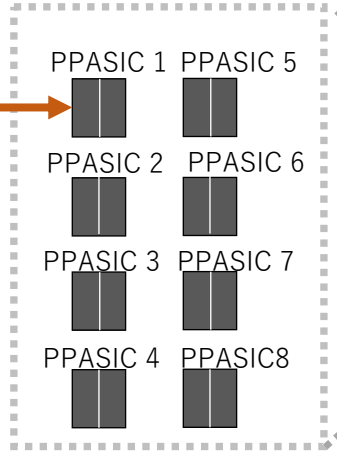
- 開発対象のエレクトロニクス＝ 「Sector Logic(SL)」
  - 実験室とは別室にあるバックエンドエレクトロニクス
  - 1枚のSLが担う領域は1/24セクター
- 検出器からSLまでにある前段回路
  - ASD　　：デジタル化、1チャンネル＝1bit
  - PSボード　：タイミング同期、40MHzのバンチ交差のうちどれ由来か
- Sector Logic1枚につき128 bit × 58 link のデータがイベント毎に送信される
  - SLは29枚のPSボードから58本の光ファイバーによりTGC Big Wheelのヒット情報を受け取る
  - 1本の光ファイバーが最大128チャンネルを取り扱う
  - 手作業で開発用のテスト入力パターンを作るのは大変難しい

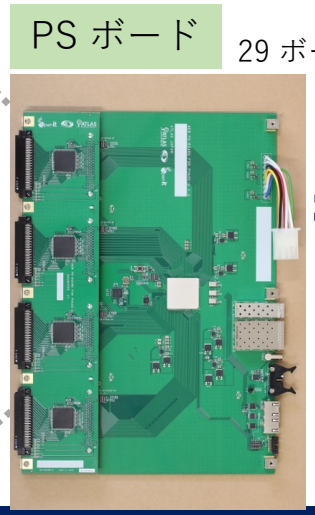




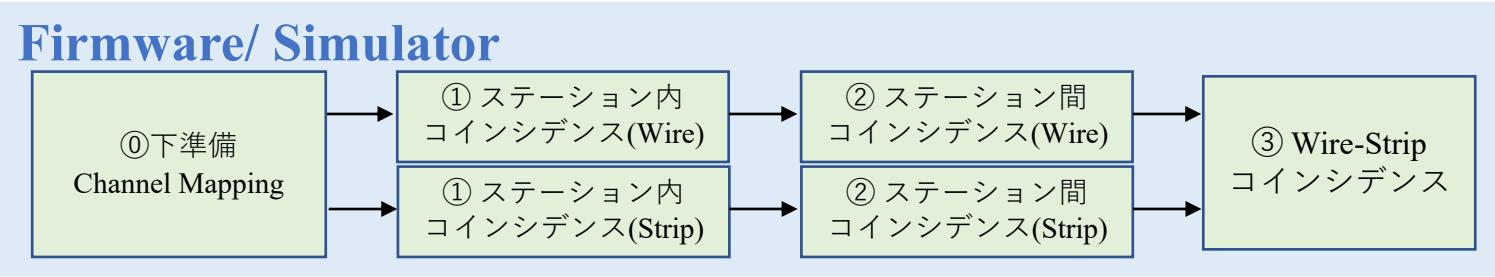








SLのトリガー系の7層コインシデンスの計算



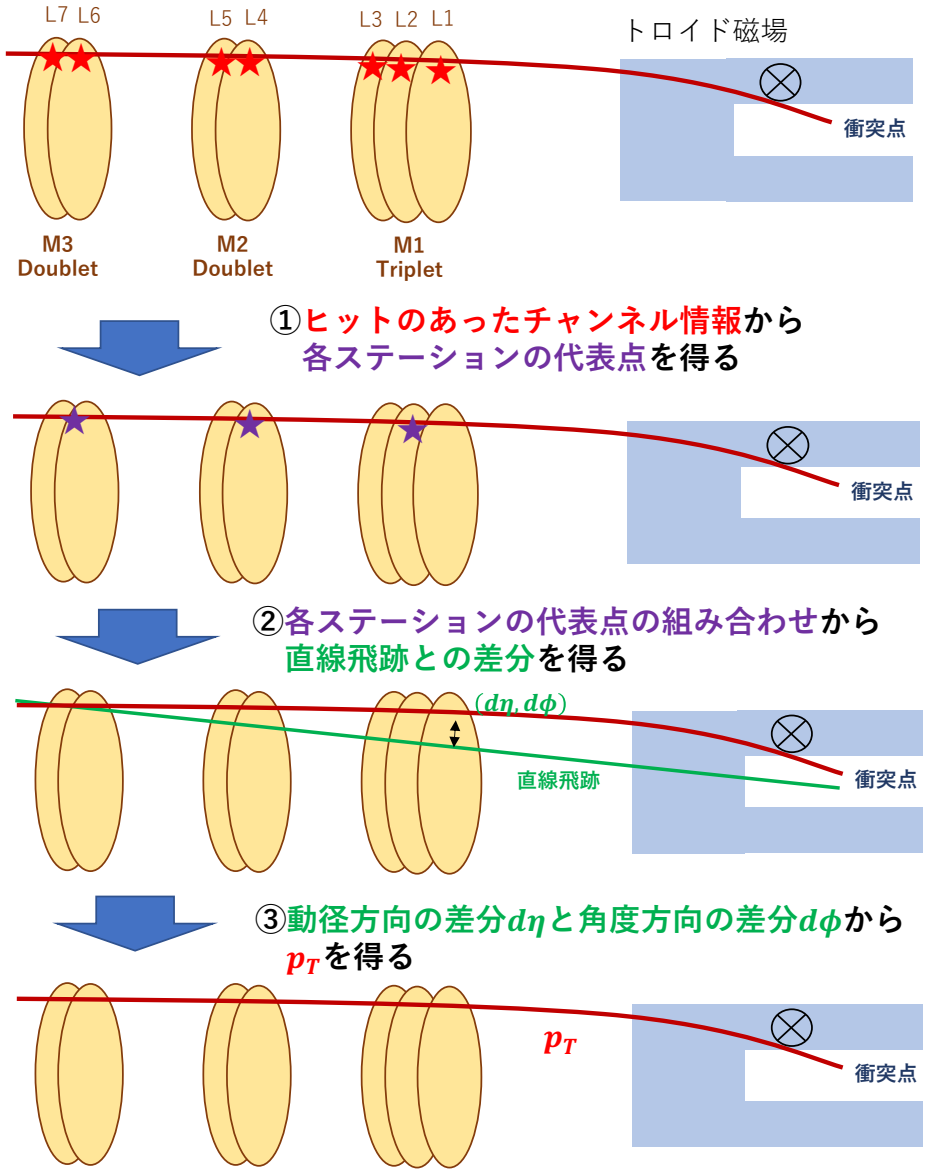
- SLトリガー系では多段の計算によってp_Tを得る
  - 機能ごとのロジックの塊＝「モジュール」と呼称

⓪　下準備：Channel Mapping
- SLがシリアルリンクで受け取ったビットマップ(128 bit × 58 link) を
  コインシデンスのためのロジカルなチャンネルに変換する

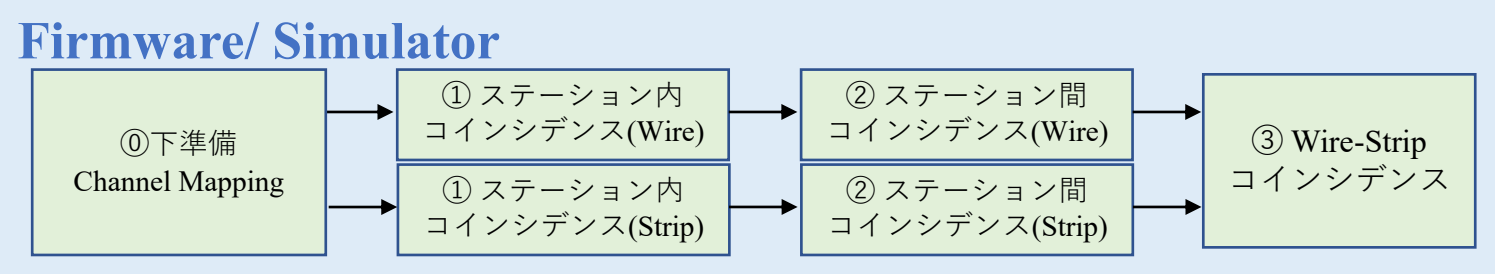
①　ステーション内コインシデンス
- チャンネルに対してコインシデンス論理をかけ、各ステーションの代表点を得る

②　ステーション間コインシデンス
- LUT(Look up table)によって直線飛跡との差分$(\Delta\eta, \Delta\phi)$を得る
  - 入力：M1・M2・M3の代表点3つの組み合わせ
  - 出力： Wireでは$\Delta\eta$, Stripでは$\Delta\phi$
- Wire、Stripを独立に計算

③　Wire-Strip コインシデンス
- $(\Delta\eta, \Delta\phi)$からLUTによってp_Tを得る
  - 入力：Wireの$\Delta\eta$＋ Stripの$\Delta\phi$
  - 出力： p_Tの閾値（「5GeV以上」などの4段階）



SLのトリガー系の7層コインシデンスの計算



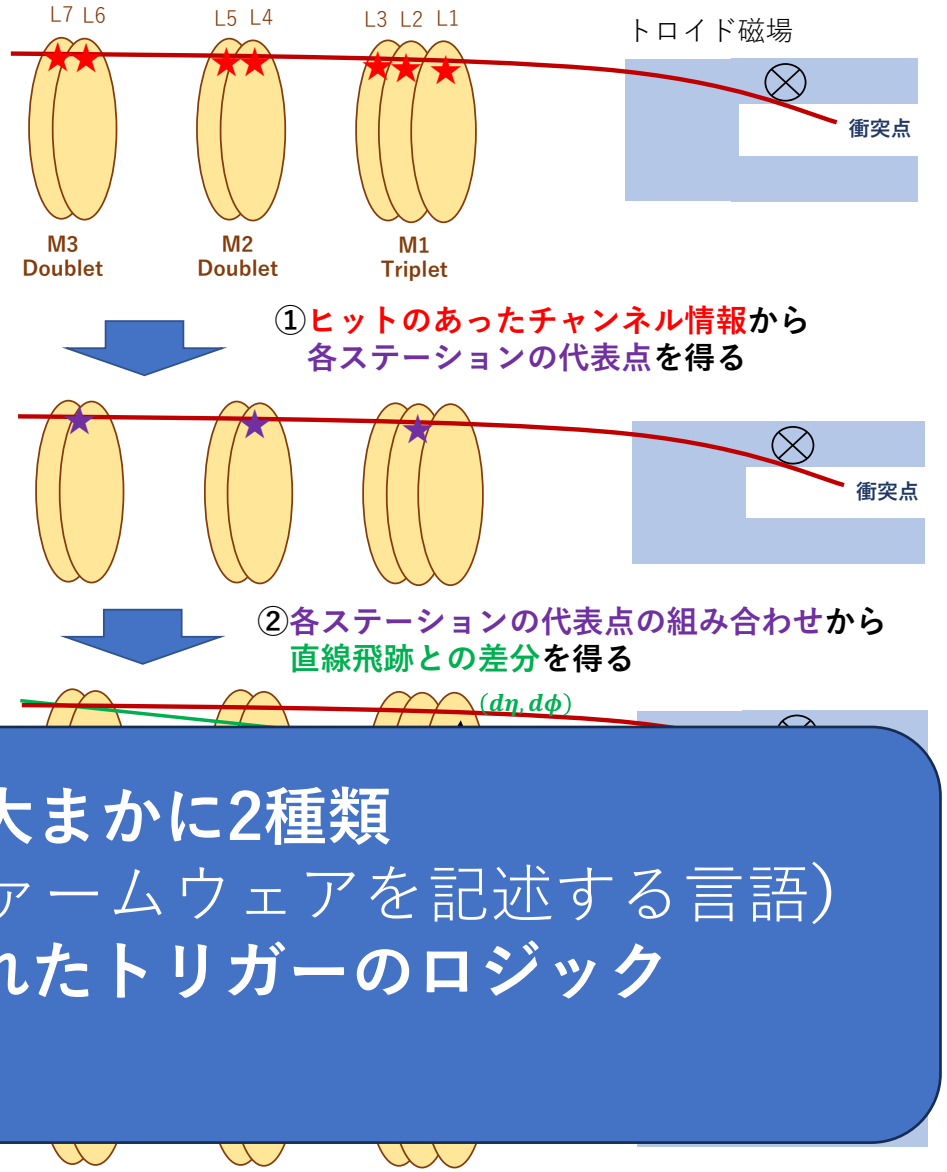
- SLトリガー系では多段の計算によってp_Tを得る
  - 機能ごとのロジックの塊＝「モジュール」と呼称

⓪　下準備：Channel Mapping
- SLがシリアルリンクで受け取ったビットマップ(128 bit × 58 link) を
  コインシデンスのためのロジカルなチャンネルに変換する

①　ステーション内コインシデンス
- チャンネルに対してコインシデンス論理をかけ、各ステーションの代表点を得る

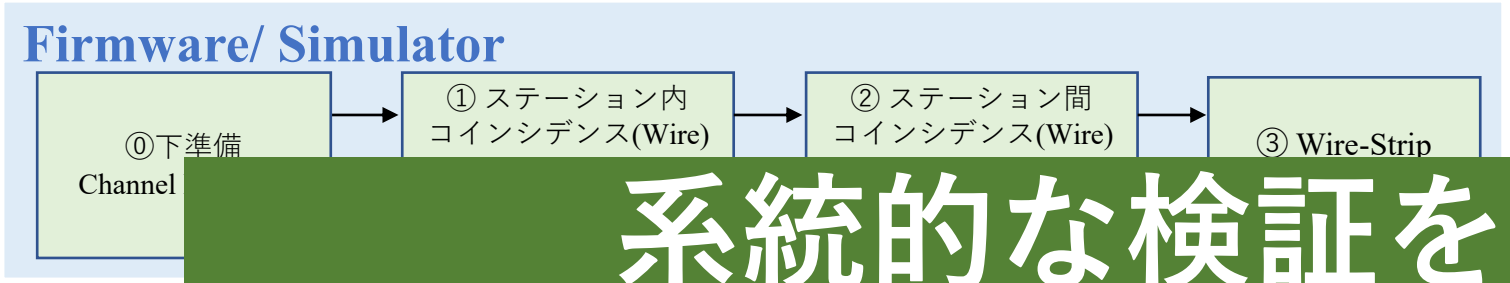
②　ステーション間コインシデンス
- LUT(Look up table)によって直線飛跡との差分$(\Delta\eta, \Delta\phi)$を得る
  - 入力：M1・M2・M3の代表点3つの組み合わせ
  - 出力： Wireでは$\Delta\eta$, Stripでは$\Delta\phi$
- Wire、Stripを独立に計算

③　Wire-Strip コインシデンス
- $(\Delta\eta, \Delta\phi)$からLUTによってp_Tを得る
  - 入力：Wireの$\Delta\eta$＋ Stripの$\Delta\phi$
  - 出力： p_Tの閾値（「5GeV以上」などの4段階）



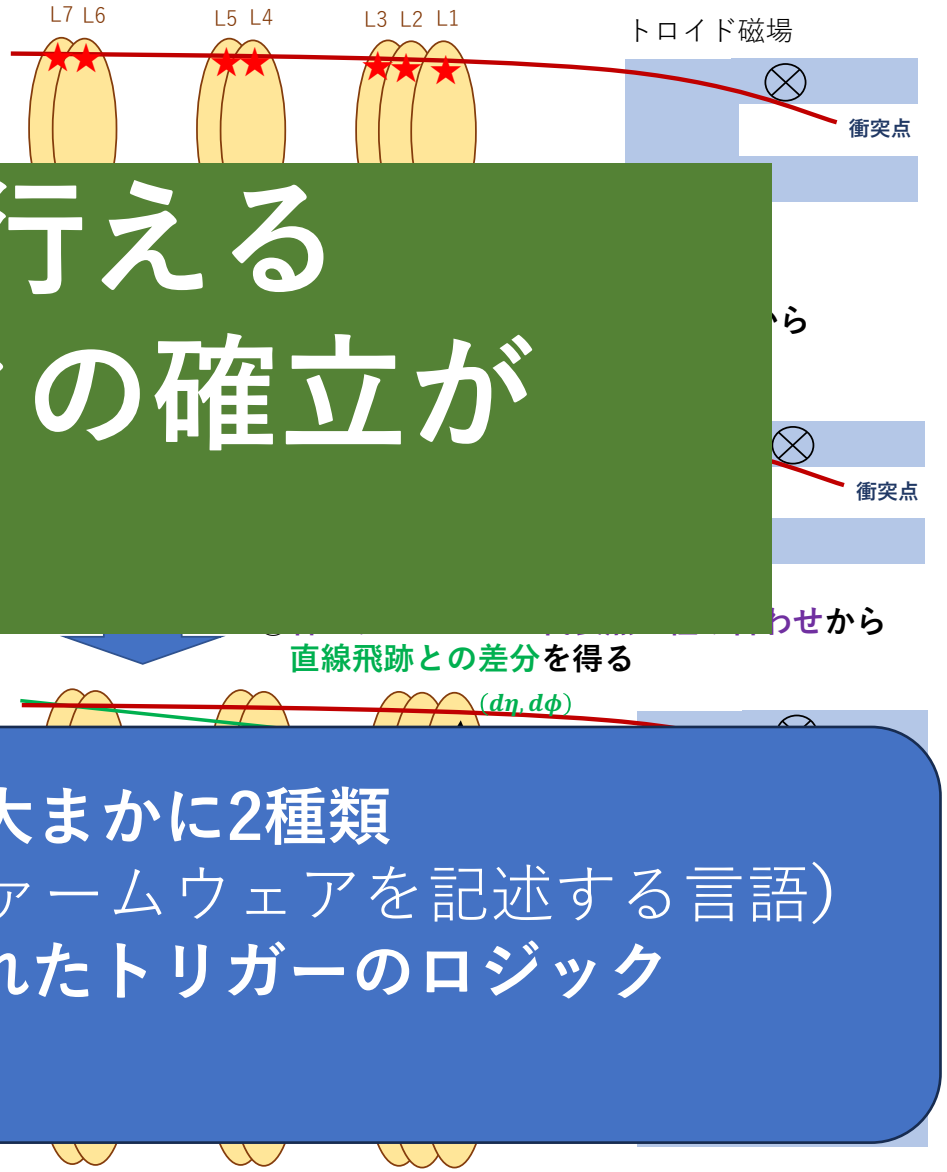
検証対象は大まかに2種類
- **HDL（ファームウェアを記述する言語）で実装されたトリガーのロジック**
- **LUT**

SLのトリガー系の7層コインシデンスの計算



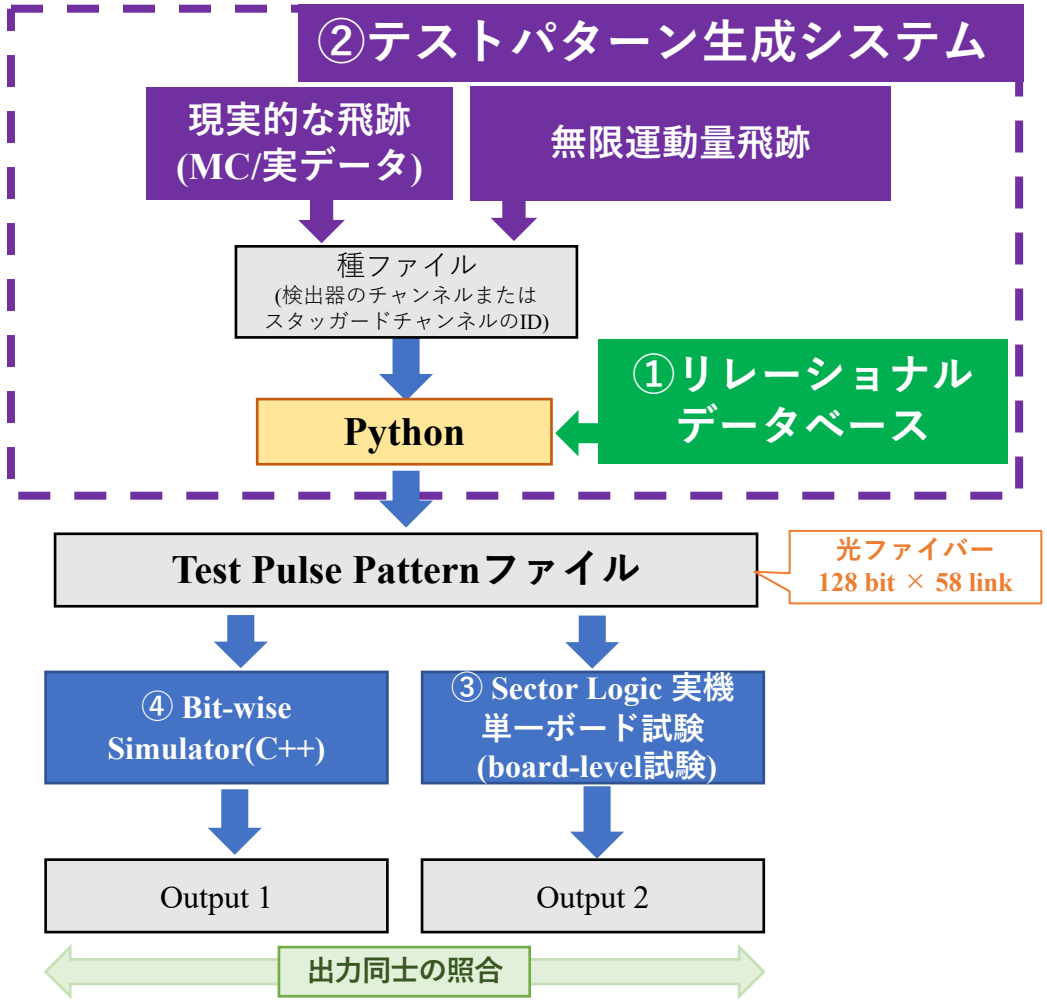
系統的な検証を行える
開発フレームワークの確立が
成功の鍵！

- SLトリ
  - 機
- ⓪ 下準
  - S
- ① ス
  - チャンネルに対してコインシデンス論理をかけ、各ステーションの代表点を得る
- ② ステーション間コインシデンス
  - LUT(Look up table)によって直線飛跡との差分$(\Delta\eta, \Delta\phi)$を得る
    - 入力：M1・M2・M3の代表点3つの組み合わせ
    - 出力： Wireでは$\Delta\eta$, Stripでは$\Delta\phi$
  - Wire、Stripを独立に計算
- ③ Wire-Strip コインシデンス
  - $(\Delta\eta, \Delta\phi)$からLUTによってp_Tを得る
    - 入力：Wireの$\Delta\eta$＋ Stripの$\Delta\phi$
    - 出力： p_Tの閾値（「5GeV以上」などの4段階）



検証対象は大まかに2種類
- **HDL（ファームウェアを記述する言語）で実装されたトリガーのロジック**
- **LUT**

SL検証機構の全体設計



コンセプト：ソフトウェアとハードウェアの相互検証による堅固な検証機構

- Sector Logicボードの実機試験とシミュレータ(C++)に共通のテストパターンを入力し、自由自在に相互検証を行うことにより、実機の開発・デバッグを進める

1. リレーショナル・データベース
   - ケーブリングやコンポーネントに関する様々な情報を一元管理
2. テストパターン生成システム
   - 無限運動量飛跡やMCデータを利用した現実的な飛跡など、さまざまなテストパターンを作成可能
   - リレーショナル・データベース内のケーブリング情報を利用
3. SL実機の単一ボード検証システム
   - ボード単体でのトリガー回路の試験フレームワークを実現
4. Bit-wise simulator
   - SLトリガー系のファームウェアと完全に同一のロジック・入出力
   - 実機出力と比較するための期待出力を生成

- シミュレーションから実機試験まで、系統的に動作検証を行うことができる研究基盤を構築した

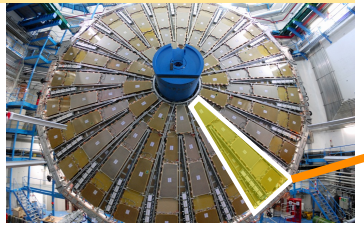
①リレーショナルデータベースの構築と利用

- SLテストパターン生成機構・SLトリガー系ファームウェア開発のために、以下の情報が必要
  - TGC検出器からSLまでのケーブリング情報
  - 検出器情報の送受信におけるデータフォーマットの情報
- Phase2 アップグレードのインフラとして幅広く応用可能な形でデータベースを設計したい

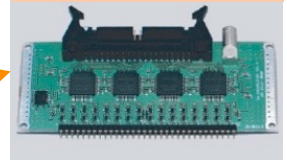
MySQLを使ってリレーショナル・データベースを作成

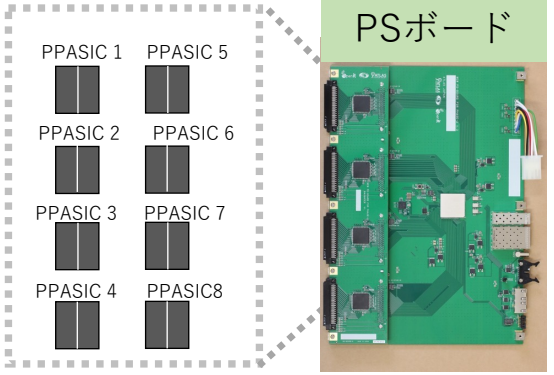
物理的な
(η, ϕ)座標

検出器チャンネル



ASDボード
1 枚で最大16ch

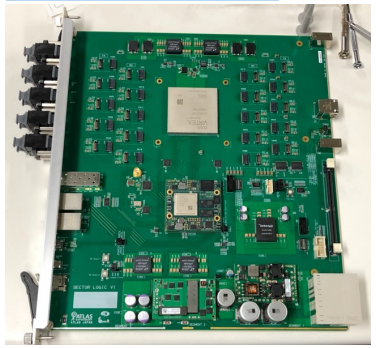




PSボード

光ファイバー
128 bit × 58 link

Sector Logic



Staggered (4-593)	レイヤー3通し番号 (1-196)	レイヤー2通し番号 (1-196)	レイヤー1通し番号 (1-196)
⋮	⋮	⋮	⋮
590	196	196	195
589	196	195	195
588	195	195	195
587	195	195	194

レイヤー1通し番号 (1-196)	ASDチャンネル (0-15)	ASD名
196	7	EW0-1
195	6	
194	5	
⋮	⋮	
189	0	EW
188	15	EW

ASD名	Subsector (E/F, phi)	FPGA番号 (1-29)	PSボード上のPPASIC番号 (1-8)	Port (A/B)
	hi0	1	PP1	A
	hi0	1	PP1	
		⋮	⋮	
NC	NULL	1	PP8	A
W4-1	Endcap-phi0	1	PP8	B
⋮				
	F			B

FPGA番号 (1-29)	Link番号 (0/1)	Bank	Channel
1	0	121	9
1	1	121	11
2	0	121	6
2	1	122	8

共通する列を使って
表同士を結合

η ID – 検出器チャンネル

検出器チャンネル - ASD

ASD – PSボード

PSボード/リンク
- Bank/Channel

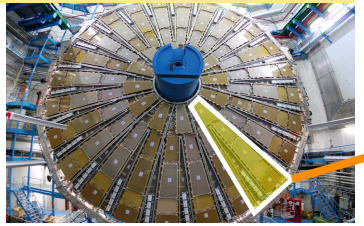
①リレーショナルデータベースの構築と利用

- 冗長性なく情報を取り扱える
  - コンポーネントの対応ごとに表を作成
  - 共通する列を使って表同士を結合
  - 配線の変更があれば該当箇所の表のみ書き換えればよい　→　情報の管理が容易！
- PythonなどとのAPIも既存のものが利用可能なため、MySQLを使用

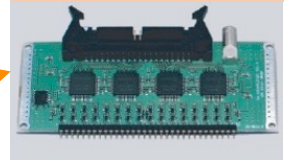
その他、
・データフォーマットを記述するデータベース
・全コンポーネントを管理するデータベース
などのデータベースも作成した

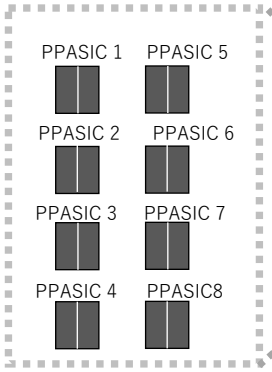
物理的な
(η, ϕ)座標

検出器チャンネル

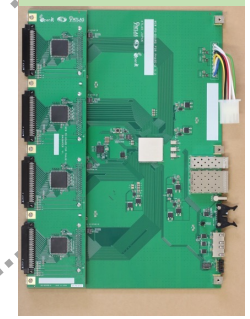


ASDボード
1 枚で最大16ch



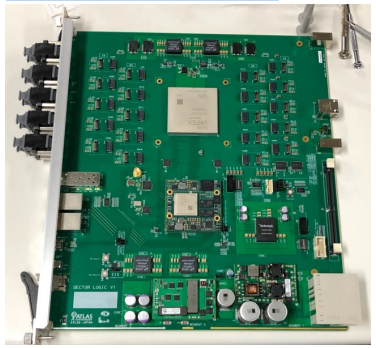


PSボード



光ファイバー
128 bit × 58 link

Sector Logic



Staggered (4-593)	レイヤー3通し番号 (1-196)	レイヤー2通し番号 (1-196)	レイヤー1通し番号 (1-196)
⋮	⋮	⋮	⋮
590	196	196	**195**
589	196	195	**195**
588	195	195	**195**
587	195	195	**194**

レイヤー1通し番号 (1-196)	ASDチャンネル (0-15)	ASD名
196	7	EW0-1
195	6	
194	5	
⋮	⋮	
189	0	EW
188	15	EW

ASD名	Subsector (E/F, phi)	FPGA番号 (1-29)	PSボード上のPPASIC番号 (1-8)	Port (A/B)
	hi0	1	PP1	A
	hi0	1	PP1	
		⋮	⋮	
NC	NULL	1	PP8	A
W4-1	Endcap-phi0	1	PP8	B
⋮				
	F			B

FPGA番号 (1-29)	Link番号 (0/1)	Bank	Channel
1	0	121	9
1	1	121	11
2	0	121	6
2	1	122	8

共通する列を使って表同士を結合

η ID – 検出器チャンネル

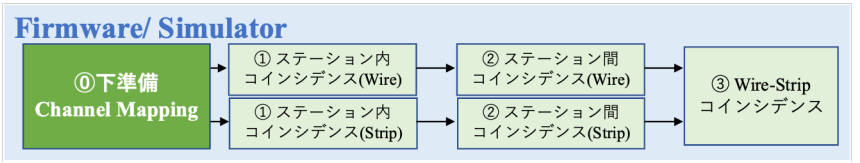
検出器チャンネル - ASD

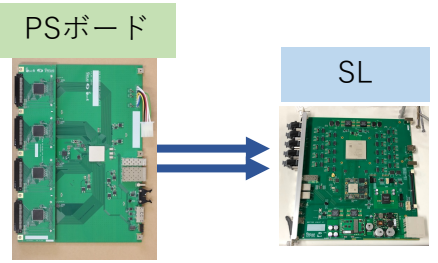
ASD – PSボード

PSボード/リンク - Bank/Channel

利用1：トリガー系ファームウェアの作成

- トリガー系下準備：Channel Mapping
  - SLトリガー系のファームウェアで最初のモジュール
  - SLがシリアルリンクで受け取ったビットマップ(128 bit × 58 link) を コインシデンスのためのロジカルなチャンネルに変換する
    - SLは29枚のPSボードから58本の光ファイバーによりTGC Big Wheelのヒット情報を受け取る
    - 1本の光ファイバーが最大128チャンネルを取り扱う





- データベースを利用してファームウェアを自動生成する仕掛けを開発した
  - ビットポジションとロジカルなチャンネルの対応関係をデータベースから取り出し、Pythonで情報を整形してファームウェア（HDLファイル）を出力した
  - SL 1枚が担う領域にある約6000のチャンネルについて、1チャンネルにつき1行でファームウェアが記述される

出力： レイヤー & チャンネル番号　　入力：リンク番号 & ビットポジション

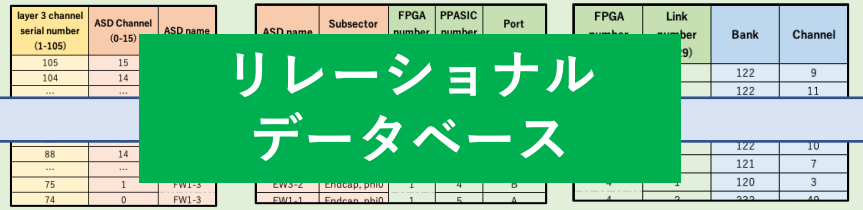
```
OUTPUT_ENDCAP_PHI0_WIRE_L3(148) <= INPUT_BITMAPS_E_phi0(6)(9);
OUTPUT_ENDCAP_PHI0_WIRE_L3(149) <= INPUT_BITMAPS_E_phi0(6)(10);
OUTPUT_ENDCAP_PHI0_WIRE_L3(150) <= INPUT_BITMAPS_E_phi0(6)(11);
OUTPUT_ENDCAP_PHI0_WIRE_L3(151) <= INPUT_BITMAPS_E_phi0(6)(12);
OUTPUT_ENDCAP_PHI0_WIRE_L3(152) <= ( INPUT_BITMAPS_E_phi0(6)(13) or INPUT_BITMAPS_E_phi0(11)(21) );
OUTPUT_ENDCAP_PHI0_WIRE_L3(153) <= INPUT_BITMAPS_E_phi0(11)(22);
```

ファームウェアの抜粋

Python

3 サブセクター × 13 レイヤー ×チャンネル番号
(Wire 7 レイヤー, Strip 6 レイヤー)

TGC検出器のチャンネルID（コインシデンスロジックへの入力）

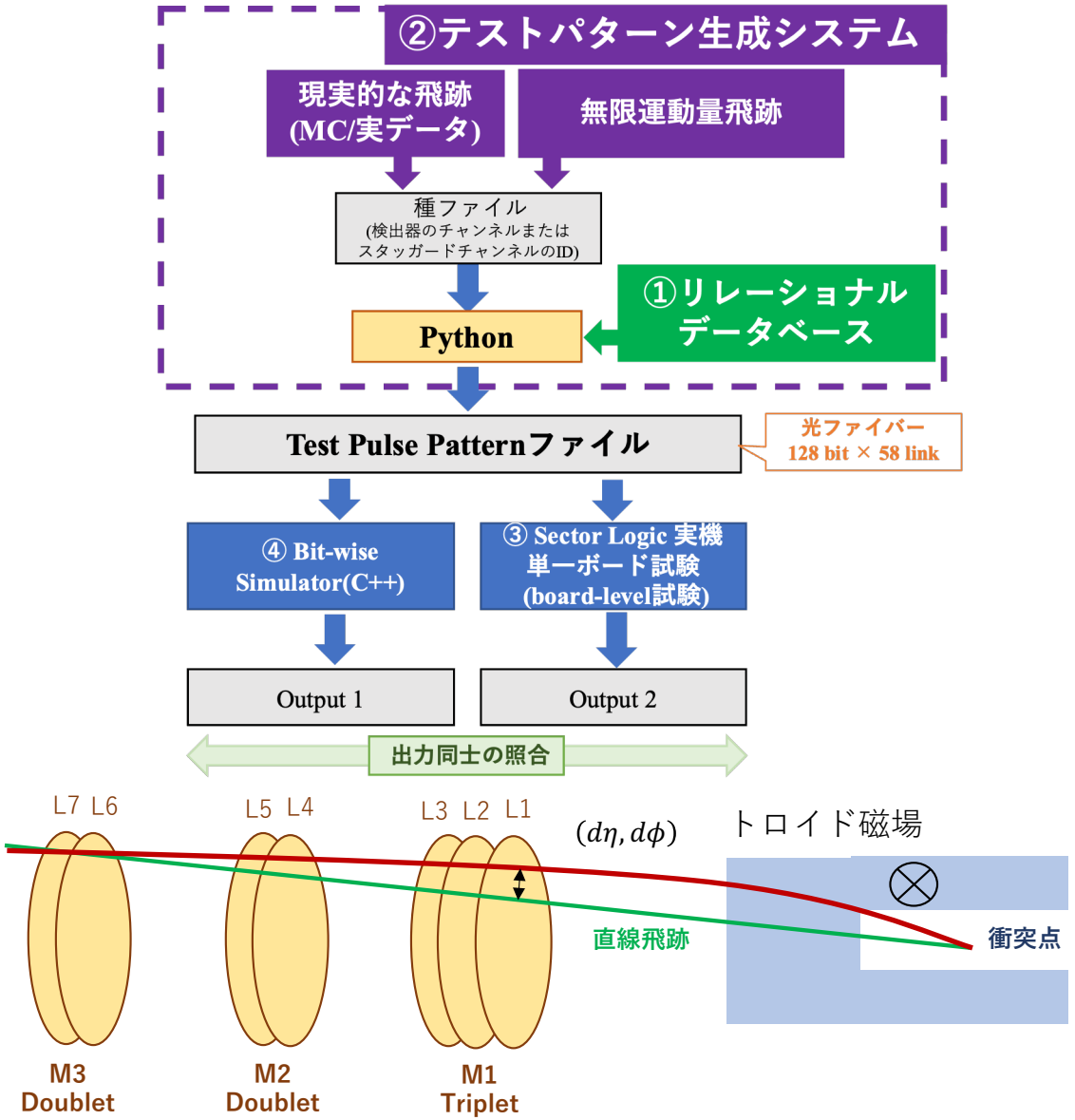


リレーショナルデータベース

光ファイバー
128 bit × 58 link

SLへのシリアルリンクのデータ形式のビットポジション

利用2：テストパターン生成システム

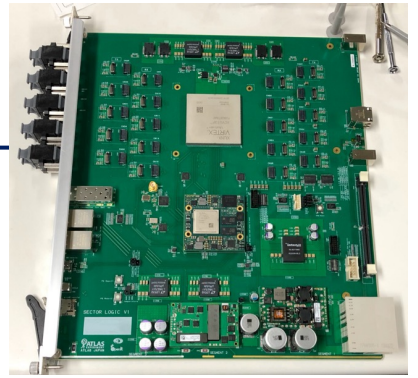


- トリガーロジックの検証のために、
  様々なテスト入力パターンを生成する機構を開発した
- 任意のヒットのパターンを、SLへの入力となる
  128bit × 58 link形式のテストベクターファイルに
  整形するための機構
  - 前述のChannel Mappingモジュールと同じく、
    テストパターンを作成するためには検出器チャンネルと
    ファイバー接続、データフォーマットの総合的な理解が必要
  - 一般的には困難な箇所だが、
    チャンネルや座標の情報とSLへ入力されるビット列の対応関係を
    リレーショナル・データベースから引き出せるため、
    自動で生成することができる！

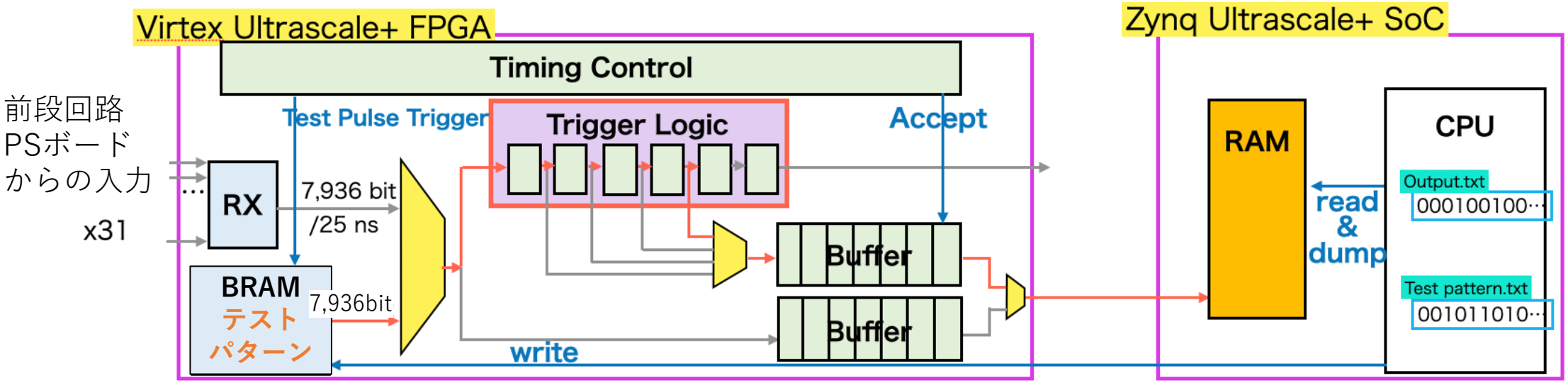
特に有用な2種類のテストベクターを実際に検証に使用

- **無限運動量飛跡**
  - 13層(Wire 7層・Strip 6層)突き抜けの直線飛跡
  - トリガーするべきトラックの中で最も単純かつ優先度の高いパターン
    - トリガーは高いp_T（=より直線に近い飛跡）のミューオンを捕まえるものなので、ロジックの最初の試験として最適
- **MCデータ/実データ**
  - 現実的な有限の曲率を持った飛跡・大統計による
    詳細なトリガーの検証

③ 単一ボード検証システム

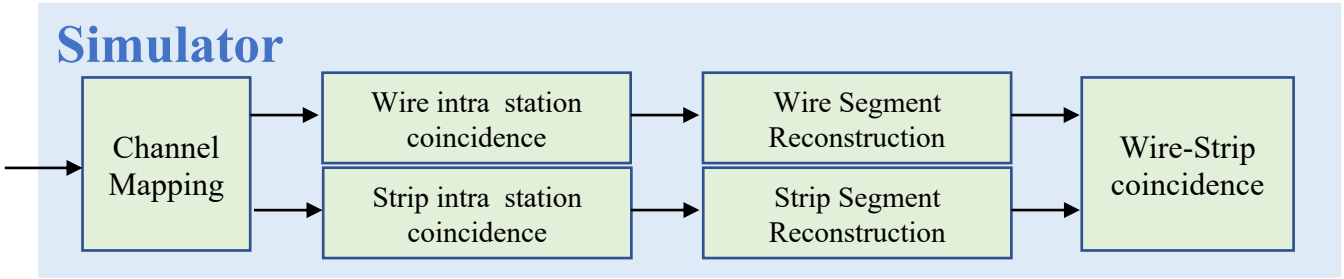


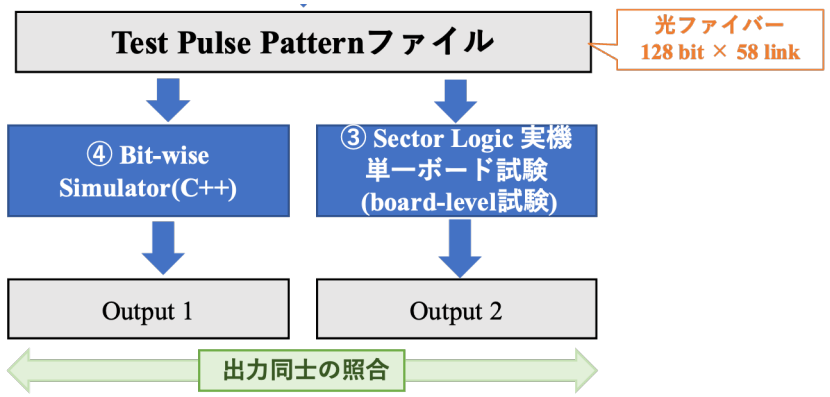
- 実機ではボード単体でのトリガー回路の試験フレームワークを実現し、
  動作検証を行なっている
  - システムはMPSoC上に確立されたLinuxによって制御される
- テストパルストリガーとBRAMでのテストパターン保存を用いて、
  全58 linkにおける同一衝突からのデータからの受信をエミュレートする機構を準備した
  - テストパターンデータ（128bit × 58 link）はFPGA上のBRAM に保存され、何度でも上書き可能
  - 読み出しの選択制
    - 入力した試験パターンに対するトリガー回路の出力、さらには多段のトリガーロジックの中間状態を柔軟に読み出しできる設計
    - 期待されない動作があったときにどの段階での不具合かを確認することができる
    - 後述のシミュレータとの比較ができるように準備してあるため、トリガーロジック自体の本質的な設計のミスなのか、
      firmwre実装時のミスなのかの切り分けが可能



④ ビットワイズシミュレータの開発

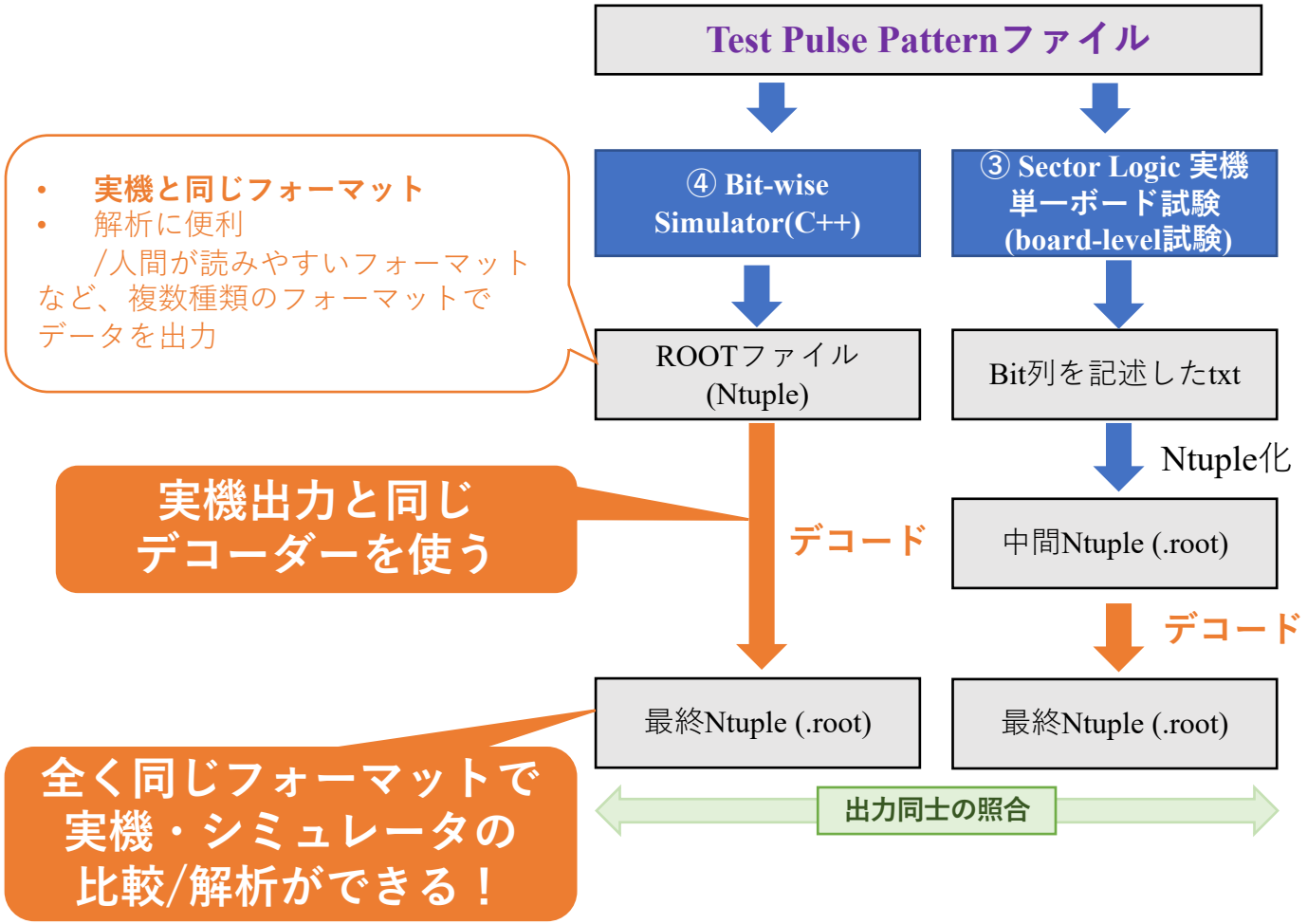
- ファームウェアのロジックを忠実に再現したシミュレータをC++で作成した
  - 開発目的1:　開発中の実機トリガー系のための期待出力を作成する
  - 開発目的2:　将来的にはATLAS検出器全体のトリガー系シミュレータの一部として使用
    - 物理解析等で使用されるシステム
- トリガーロジックのファームウェアは多段のモジュールで構成され、シミュレータはモジュールの入出力の単位で実機と完全一致
  - フロートによる近似ではなくビットによる計算を行う
  - トリガーの制約に起因するファームウェアの構造も全て一致するようにソフトウェアを実装した
- ファームウェアと同じテストパターンファイル・LUTファイルを直接使用して、トリガー出力の照合をする
  - 初期化に時間のかかるSL実機よりも素早くLUTの試験ができる
- 設定ファイルのフラグを使用して途中計算を標準出力させられるため、興味のあるイベントのトリガー計算について詳細なログを得られる
  - ログを利用してロジック・実機の応答の理解を深める！





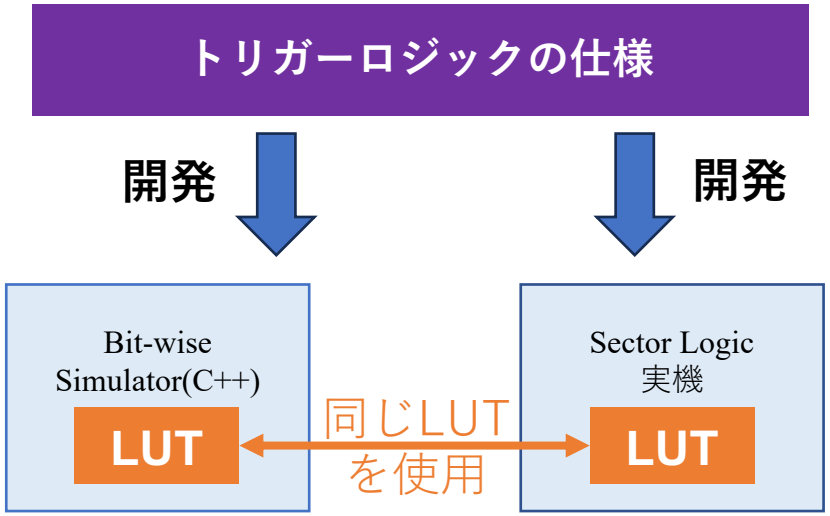
実機・シミュレータの比較のフロー

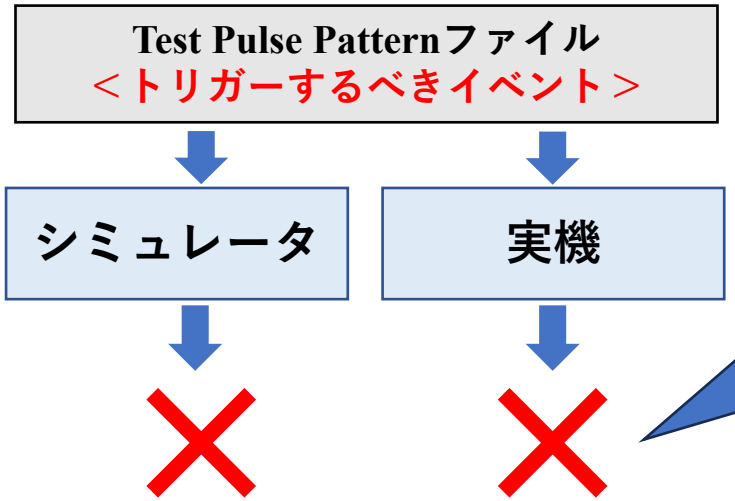
- 実機試験とファームウェアに同じテストパターンを使用し、相互検証する
- 実機
  - モジュールからの出力を
    テキストファイルに書き出す
  - Ntuple化
  - ビット列をデコードして解析用の
    ROOTファイルを生成
- シミュレータ出力
  - 複数のフォーマットの出力を用意
    - **実機と同じフォーマットのビット列**
    - 解析に便利
      /人間が読みやすいフォーマット
  - 実機の期待出力となるビット列に対して、同じデコーダーを噛ませる
- 同一のフォーマットのデータとして直接比較できる！



実機のロジックの解析・修正

- MCデータを利用した大統計によるトリガー応答の確認をした
- デバッグにあたって見たもの
  - MCで生成されたミューオンのTruthの運動量p_Tへの応答が意図した挙動になっているか
  - シミュレータと実機の応答の一致性



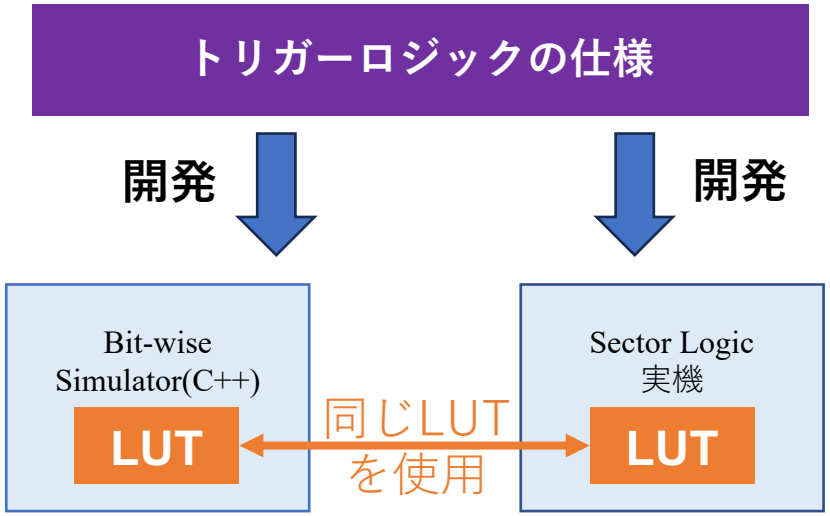


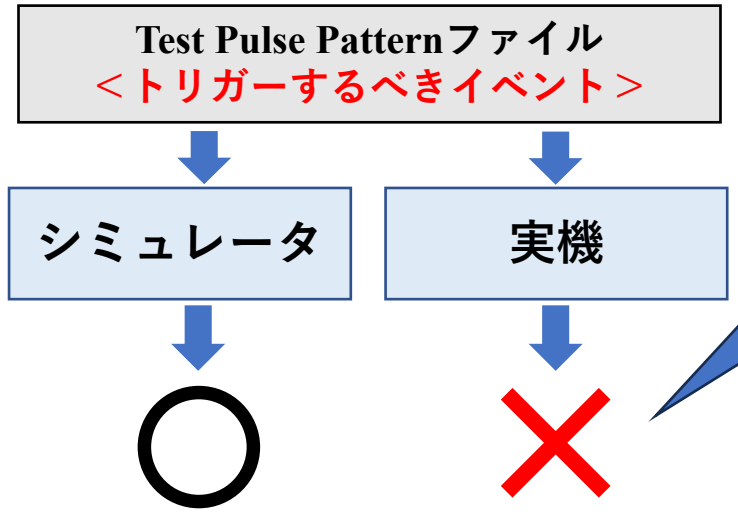
実機・シミュレータ共に
トリガー失敗した場合

シミュレータ・実機で
共通して使用している
LUTが原因の
トリガー失敗の可能性が高い

実機のロジックの解析・修正

- MCデータを利用した大統計によるトリガー応答の確認をした
- デバッグにあたって見たもの
  - MCでミューオンのTruth運動量p_Tへの応答が意図した挙動になっているか
  - シミュレータと実機の応答の一致性





実機・シミュレータの出力が食い違った場合

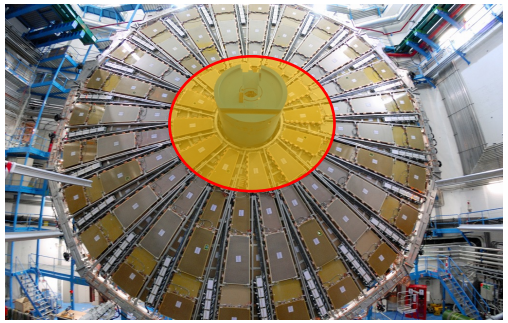
この場合は
- **実機のトリガーロジック**（ファームウェアの記述）**にミスがある**

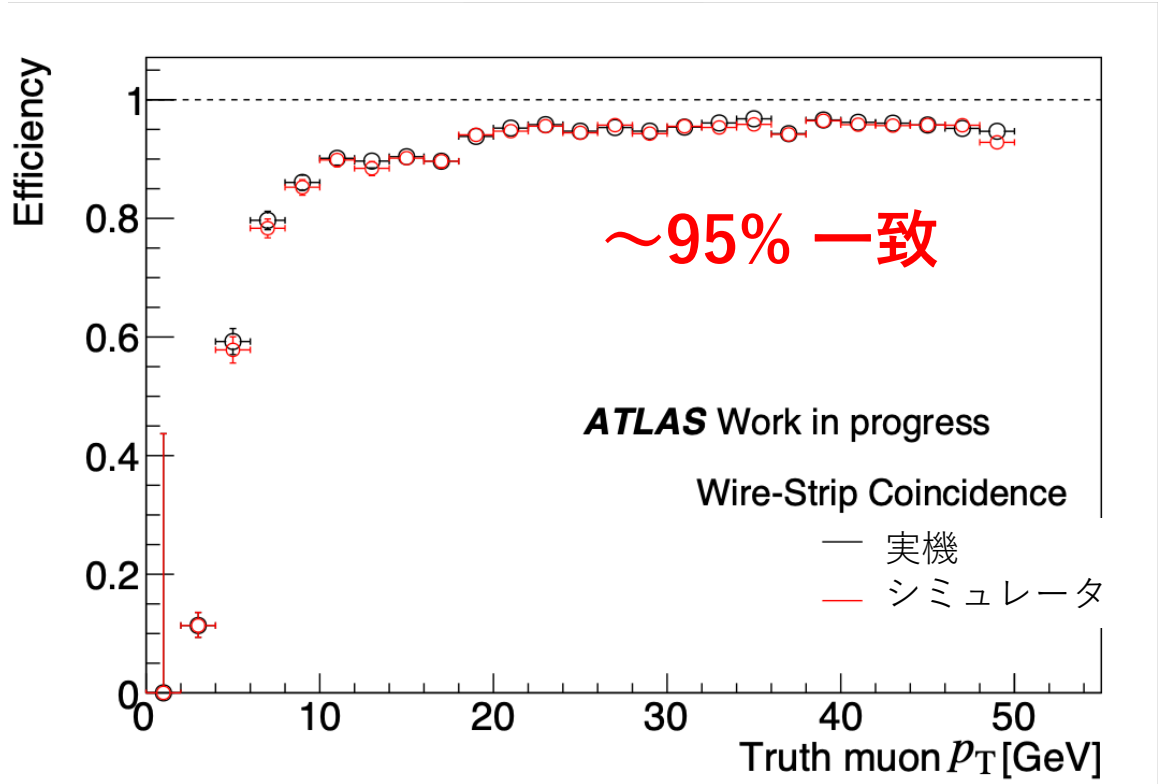
これは「トリガーすべきイベント」だが、
- **シミュレータが余分にトリガーを出してしまっている**

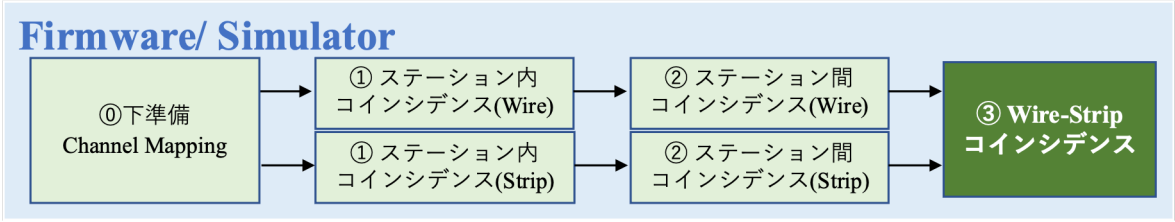
場合もある

実機・シミュレータの比較

- ビーム軸付近の領域では実機とシミュレータ出力が非常に良く一致
  - 100 K イベント, no-pileup, 0< p_{T} < 50 GeV
  - シミュレータ出力が比較対象としてよく機能している！

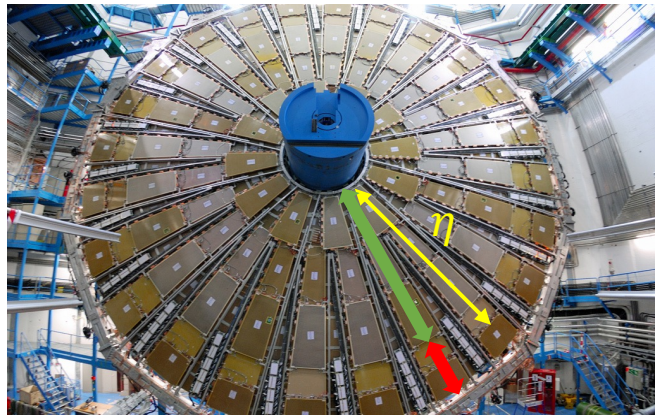


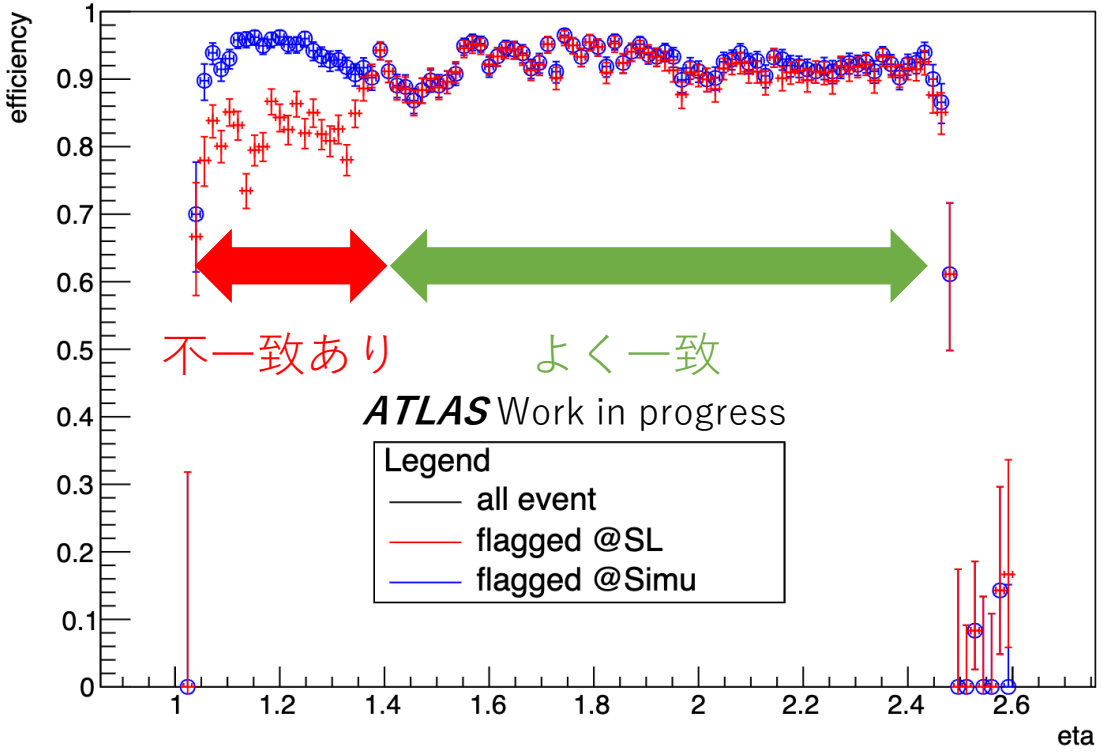


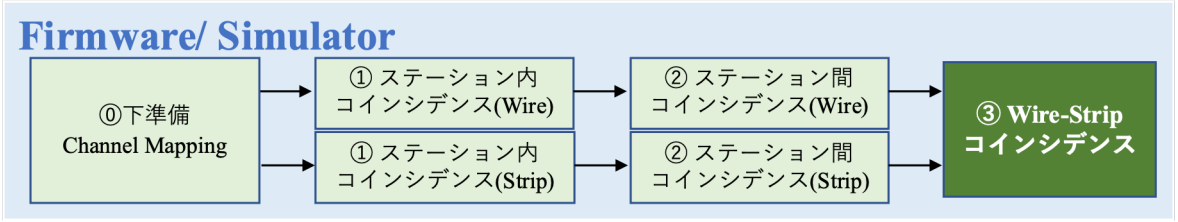


領域ごとの実機・シミュレータ比較

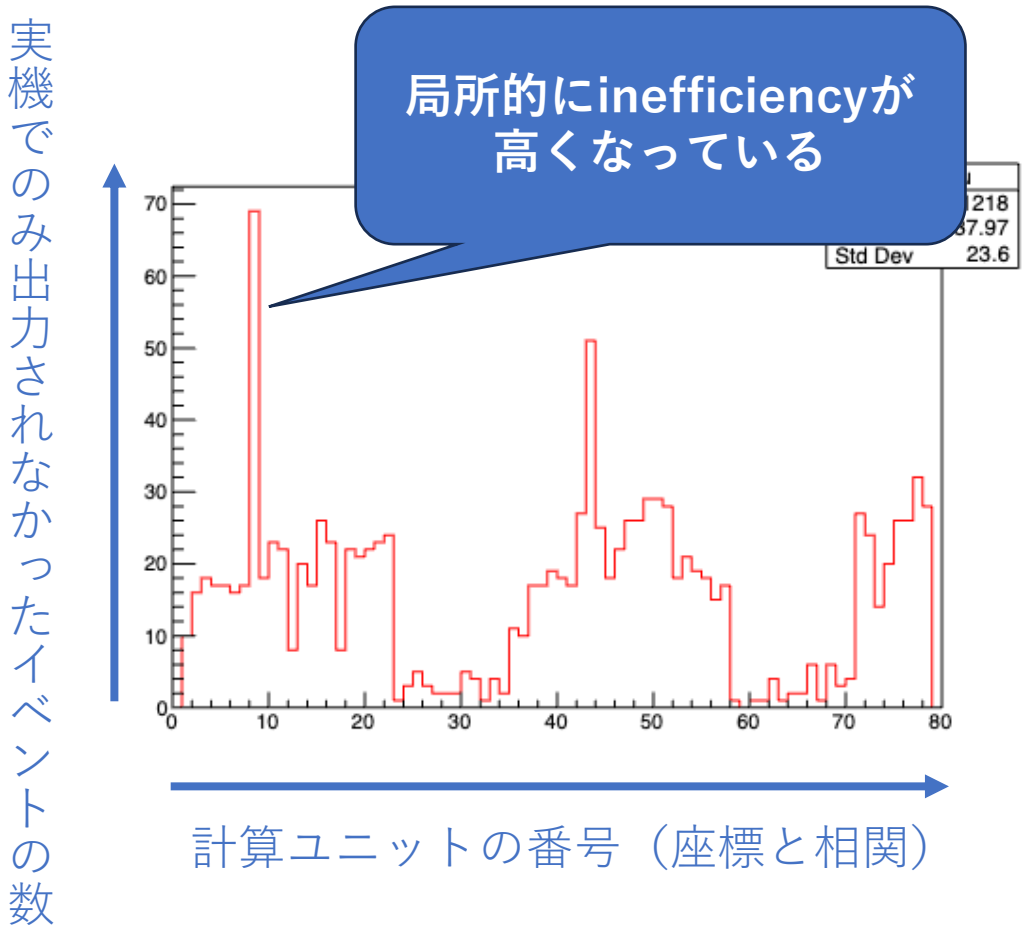
- 領域によって実機とシミュレータの出力一致率が異なる
  - 現在はデバッグの最中
- トリガーロジックの構造が大きく変化する$\eta = 1.4$を境に実機とシミュレータの出力がズレる
  - 右図は25 GeV以上のミューオンのみのイベントであるため、Flagが立つべき
  - Efficiencyの低い実機に不具合があると考えられる





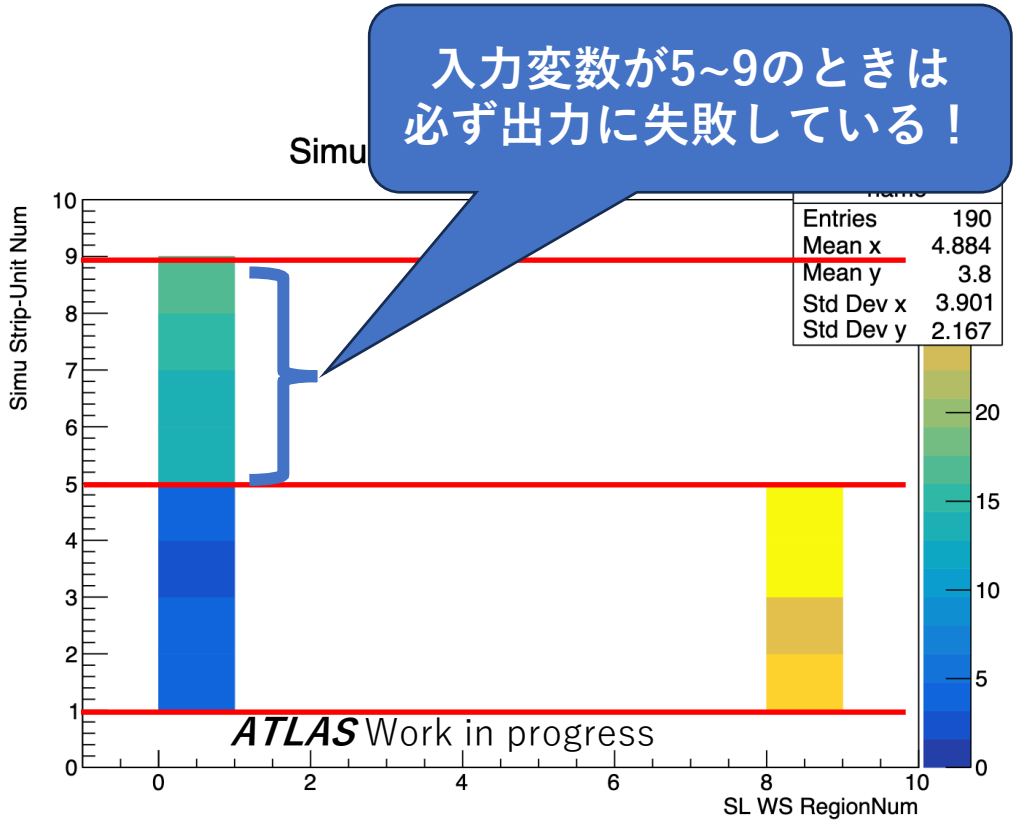


例1：実機のみ出力に失敗しているイベントの精査



- 実機でのみ出力に失敗していた
  イベントを計算ユニットごとに
  プロットした
  - トリガーは並列・高速に計算されるため、
    検出器のチャンネルごとに
    計算ユニットが分かれている
- ある計算ユニットで局所的に
  inefficiencyが高いことがわかった！
  →そのユニットについて調査

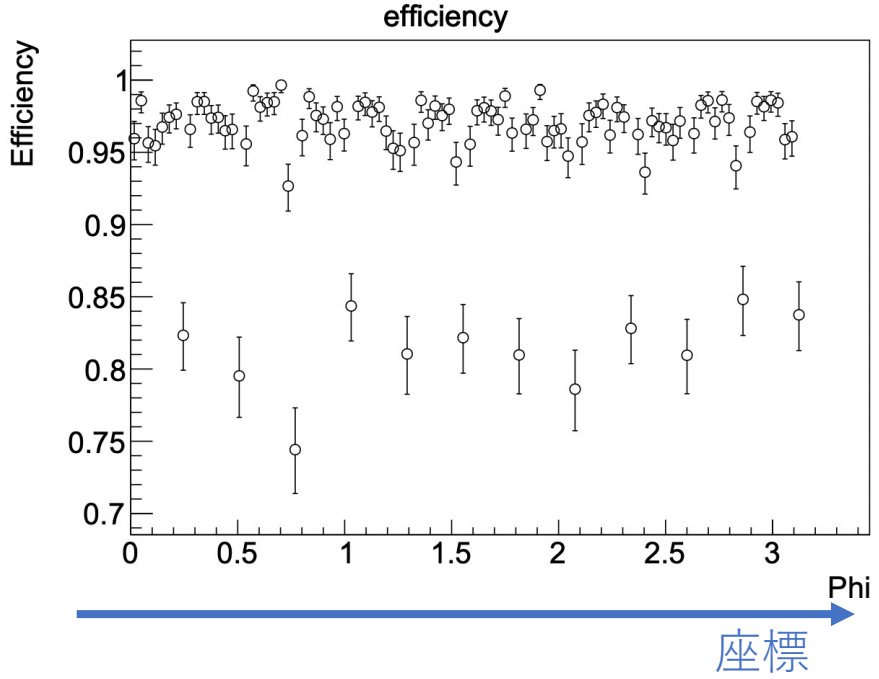
例1：実機のみ出力に失敗しているイベントの精査

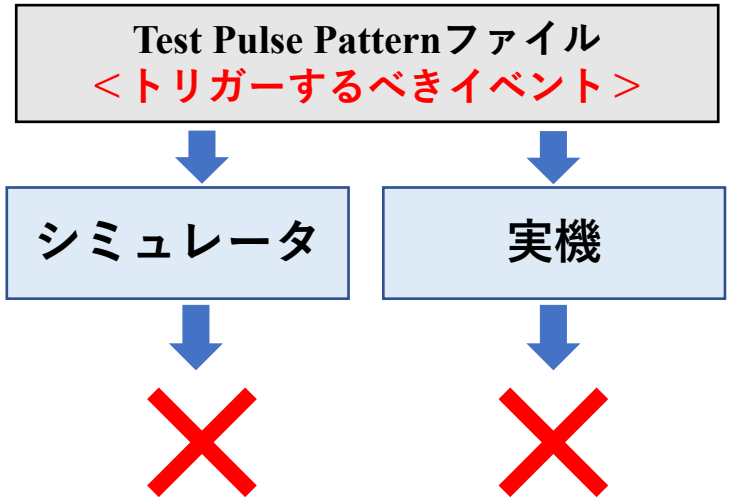


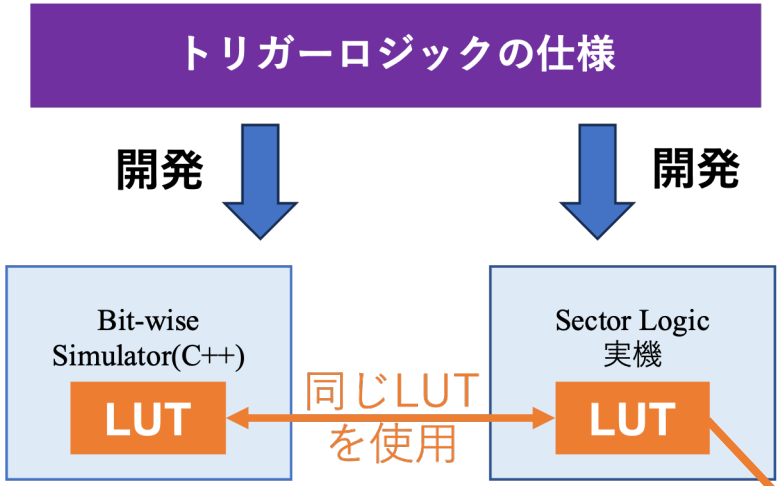
- 実機でのみ出力に失敗していた
  イベントを計算ユニットごとに
  プロットした
  - トリガーは並列・高速に計算されるため、
    検出器のチャンネルごとに
    計算ユニットが分かれている
- ある計算ユニットで局所的に
  inefficiencyが高いことがわかった！
  →そのユニットについて調査
- 実機の出力の有無と相関する変数を発見
  →デバッグするべき場所を絞りこめた！

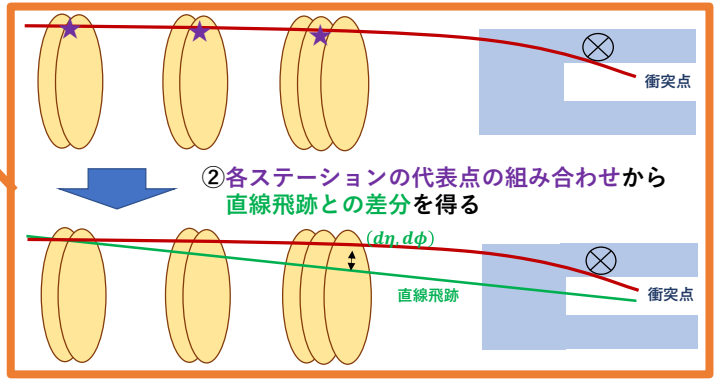
例2：LUT由来のinefficiencyの精査

- 実機・シミュレータで同じ傾向のinefficiencyが見えた
  - 座標に対して周期的
    - 周期＝1枚のSLのカバー領域
    - おそらく1点（または数チャンネルの塊）が不感領域になってしまっている
  - 実機・シミュレータで同じ傾向→LUTが原因と考えられる



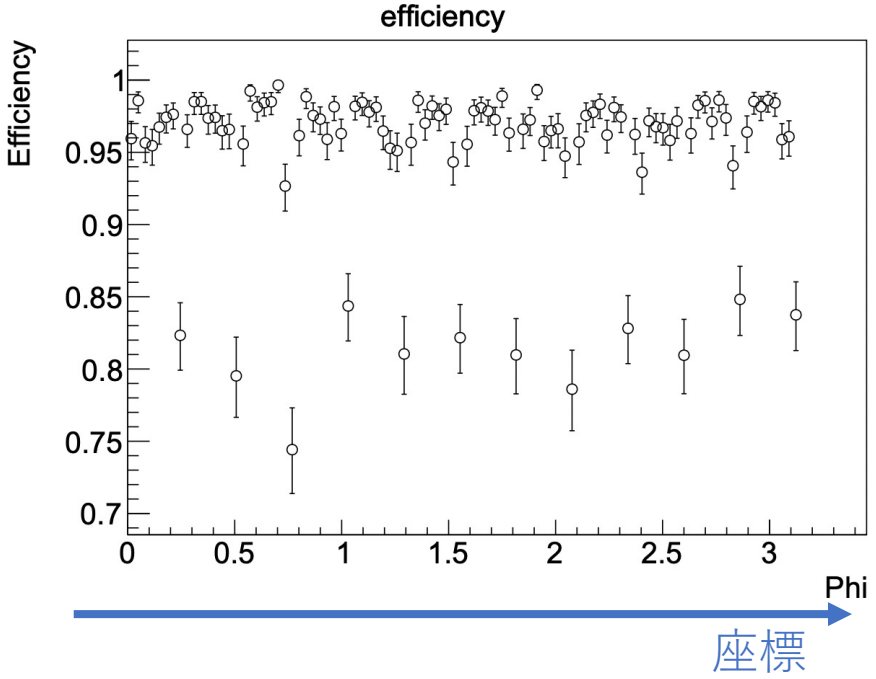






例2：LUT由来のinefficiencyの精査

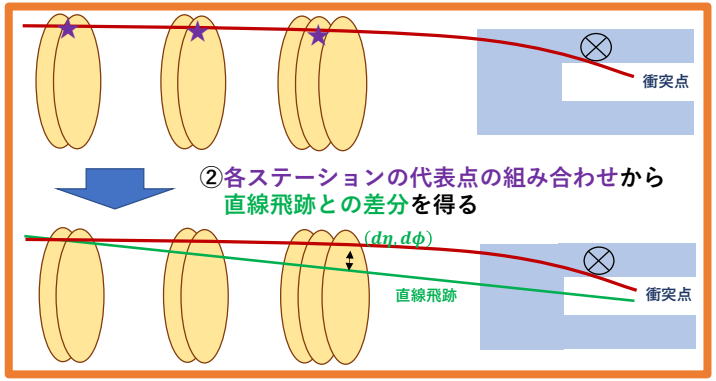
- 実機・シミュレータで同じ傾向のinefficiencyが見えた
  - 座標に対して周期的
    - 周期＝1枚のSLのカバー領域
    - おそらく1点（または数チャンネルの塊）が不感領域になってしまっている
  - 実機・シミュレータで同じ傾向→LUTが原因と考えられる
- シミュレータでトリガー計算のログを出力し、どのチャンネルの組み合わせでLUTに抜けがあるのか精査した
  - 下の画像はシミュレータで標準出力したLUTの入力と出力
- 「M3ステーションの62番目のチャンネル」を含む組み合わせがLUTに含まれていないことが確認できた！



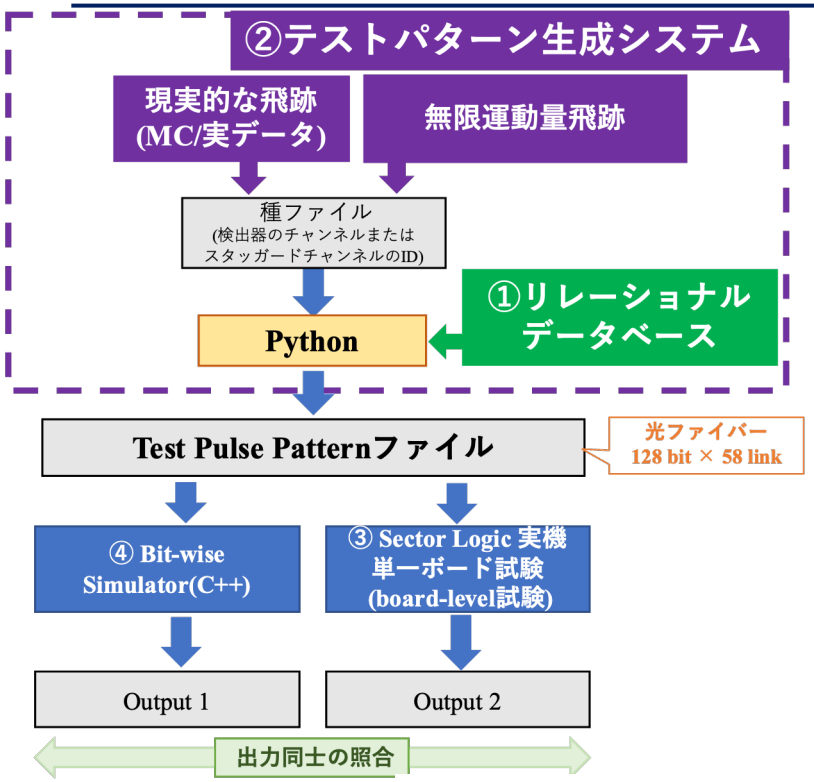

```
i_M3_PriorityNum = 0d5, M3(unit_ch) = 0d006, M3(excel) = 0d62
i_M2_PriorityNum = 0d3, M2(unit_ch) = 0d010, M2(excel) = 0d62
i_M1_PriorityNum = 0d1, M1(unit_ch) = 0d018, M1(excel) = 0d62
      M1 = 0x012,  M2 = 0x00a,  M3 = 0x006 (ID in each unit)
      EXCEL : M1 = 0d062,  M2 = 0d062,  M3 = 0d062,
        i_M1_PriorityNum = 0x001,  i_M2_PriorityNum = 0x003,  i_M3_PriorityNum = 0x005,
        chamber       = 0x0
        unit ID       = 0x3
        subunit ID    = 0x1
          Address  = 0x0032e
          key  = 0x00b2e
        MAP output    = NOT EXISTS
========== /@positionIDmaker_strip, for-loop start ==========
```

62番目の代表点

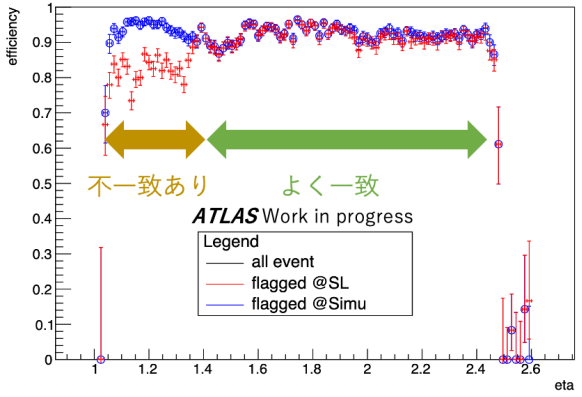
LUTの出力なし



まとめ



- HL-ATLAS実験に向けて、Phase2アップグレードが進行中
- 実機によるトリガー系の試験が進行中
  - ボード単体でのトリガー回路の試験フレームワークを実現
  - MCデータを使った大統計のテストパターンを使用
  - 開発・デバッグ
    - Truth情報との相関を使用した精査
    - シミュレータ出力との比較



今後の課題

- まだ傾向の掴めていない実機のバグが存在している
  - シミュレータの方が詳細な情報を出力できるため、
    それらの情報との相関も見ながらデバッグを進めたい