高輝度 LHC-ATLAS 実験初段ミューオントリガーにおける MPSoC と FPGA を用いた Rust による制御システムの開発

東京大学 大坪 航

2025-11-17 月 計測システム研究会 2025@J-PARC

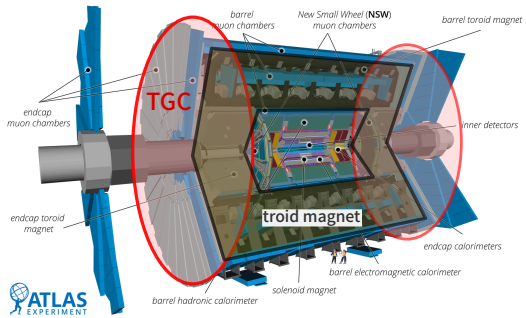
Outline

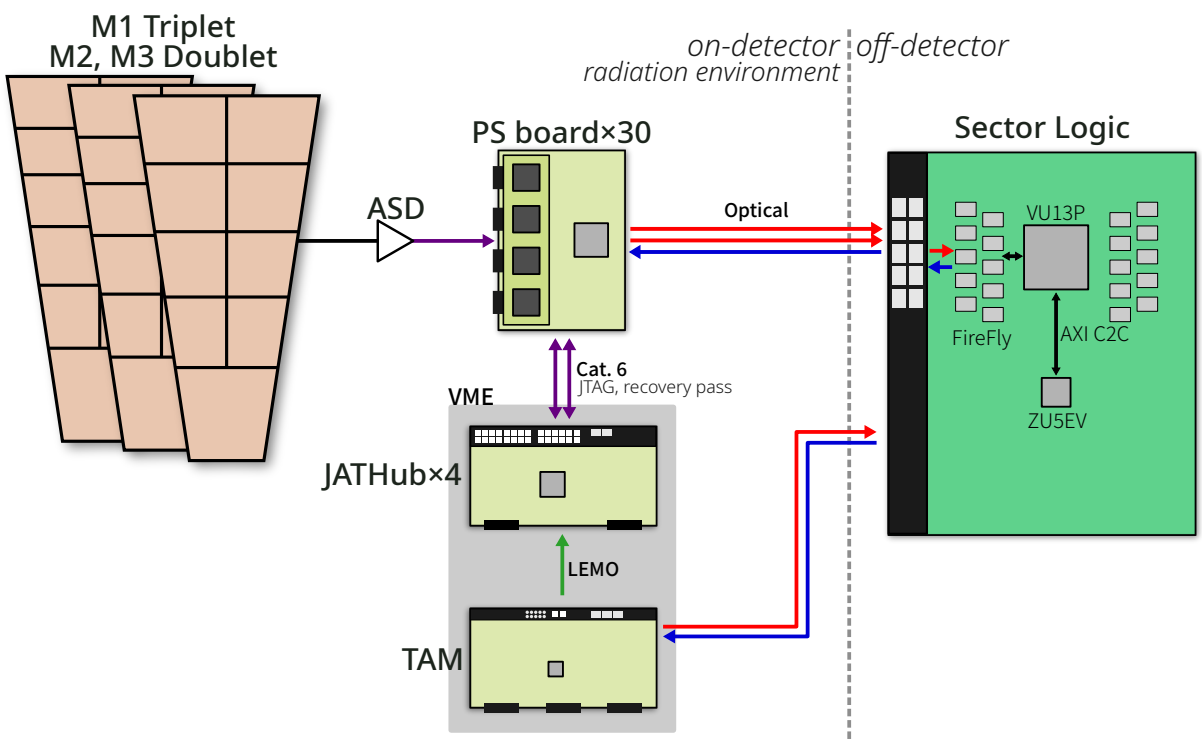
主題:
Rust を用いた SoC における制御ソフト実装の技術的詳細

目次

HL-LHC ATLAS ミューオントリガーエレクトロニクス

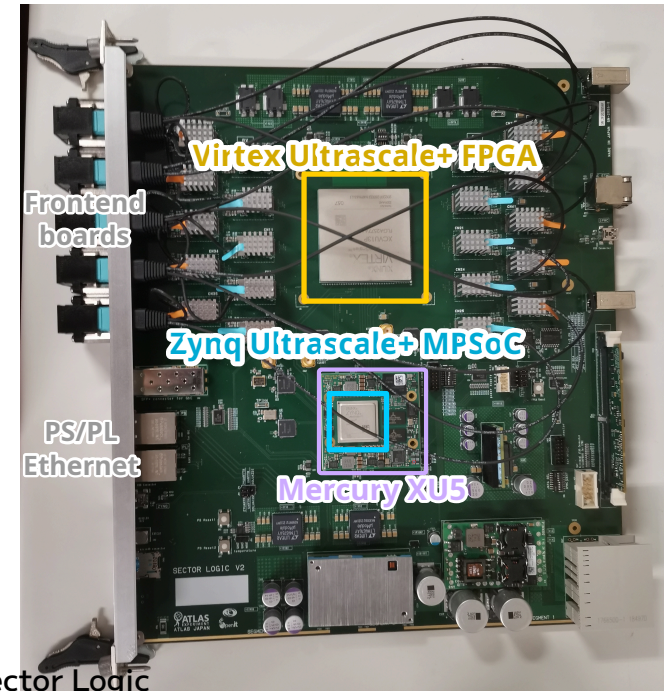
- Level-0 ミューオントリガーではオンラインでミューオンのp_Tを再構成
- HL-LHC に向けてエレキをアップデート
- 主要な構成要素
  - バックエンド: **Sector Logic(SL)**
  - フロントエンド: PS board, JATHub, TAM
- 各エンドキャップを24セクターに分割。計 48 枚の SL で構成。
- SL の役割
  - トリガー、ヒットデータ読み出し、監視・制御
  - FPGA 上のロジックおよびフロントエンド含めたボードが対象



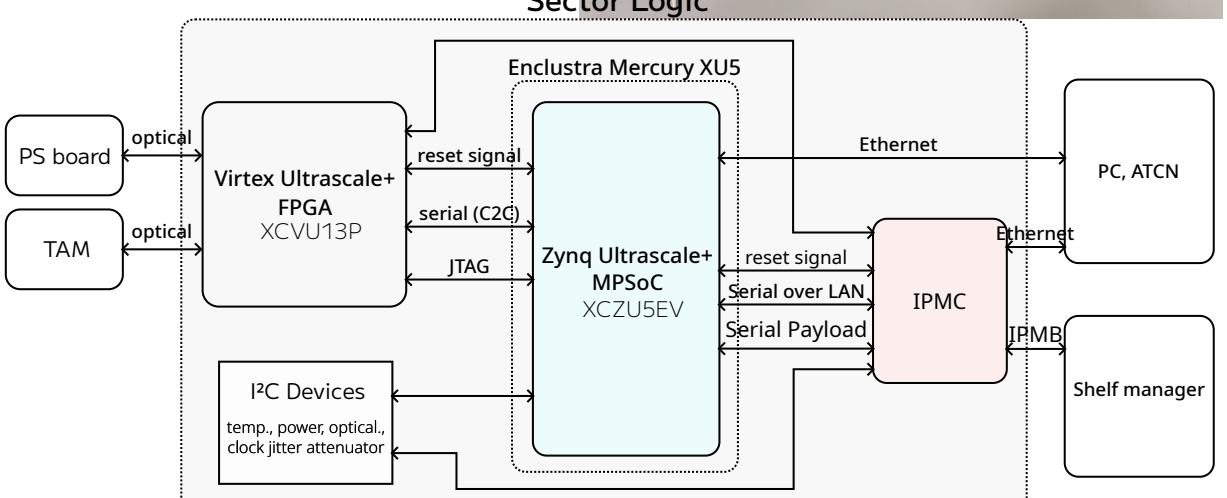


セクターロジック

- ATCA 規格に準拠
- **Virtex UltraScale+ FPGA(VU13P)**
  - 4 つの Super Logic Region(SLR)から構成
  - 20 個の FireFly 光モジュールで他のボードと接続
  - トリガー・読み出しを担う
- **Zynq UltraScale+ MPSoC(ZU5EV)**
  - メザニンボード(Enclustra Mercury XU5)上にある
  - APU (64bit ARM) + RPU + PL
  - イーサーネットあり
  - Linux が動く
- I²C 素子
  - センサー系: 温度、電源
  - スローコントロール:
    - FireFly
    - クロックジッタークリーナー
- IPMC: IPMI 規格によるボードマネジメント



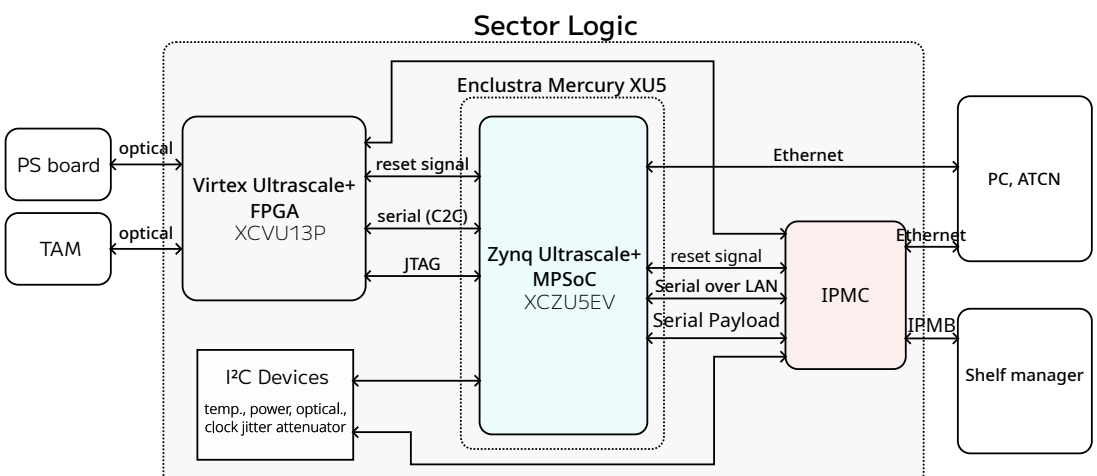
Sector Logic



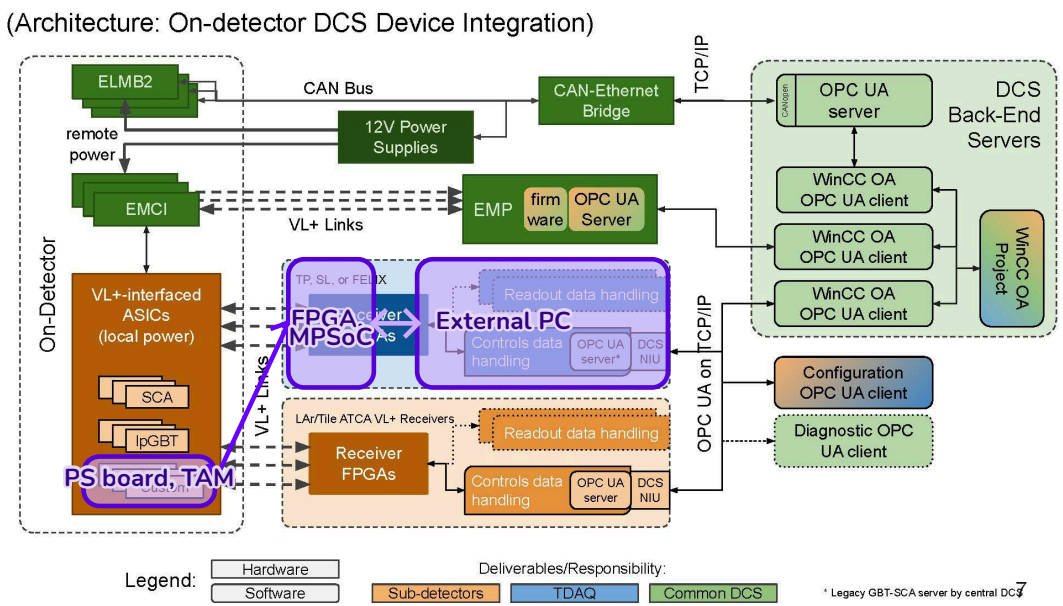
目次

制御システム概要

- MPSoC は外部への Ethernet 接続あり
  - 従って ATLAS 全体の制御システムと通信可能
- MPSoC は
  - SL の I²C 素子と通信できる
  - FPGA(VU13P)と AXI Chip2Chip で接続している
- FPGA(VU13P)は AXI Chip2Chip に繋がったレジスタがある
  - トリガー/読み出しの制御
  - フロントエンドボードの制御
- ⇒ **MPSoC を起点としたソフトで制御する**

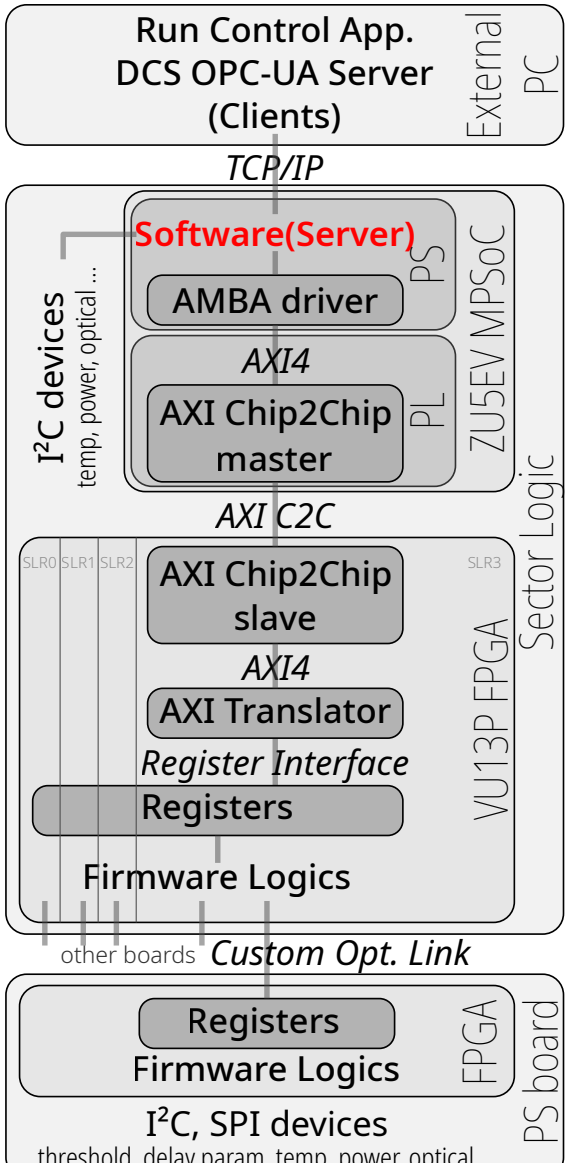


紫の部分が責任のある範囲



制御パス詳細

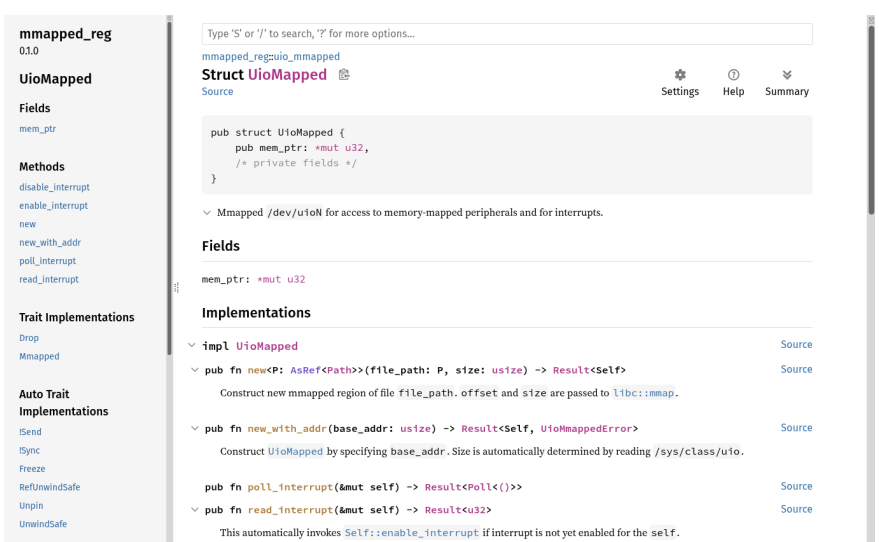
- システムを 2 つに分割(MPSoC および外部 PC)
  - MPSoC: 直接ハードを制御するサーバーを実行
    - **唯一**のゲートウェイとする
  - 外部 PC は ATLAS のシステムと接続されたクライアントを実行
  - 2 者は TCP/IP で接続
- MPSoC の PS APU で Linux 実行
  - カーネルや MPSoC 上の各素子のファームウェアなど: Petalinux
  - ルートファイルシステム: AlmaLinux
    - `dnf`のラッパーで作成
- PL-PS は AXI(`M_AXI_HPM{1,0}_FPD`, `M_AXI_HPM0_LPD`)
  - Linux 上では UIO からアクセス(memory-mapped)
- PL に AXI IP が複数あり
  - AXI Chip2Chip, AXI GPIO, System Management Wizard,...
- MPSoC-FPGA 間は AXI Chip2Chip (with Aurora 64B/66B)
- FPGA 内部は Register Interface
  - SLR 間通信用のカスタムプロトコル



設計・開発の戦略

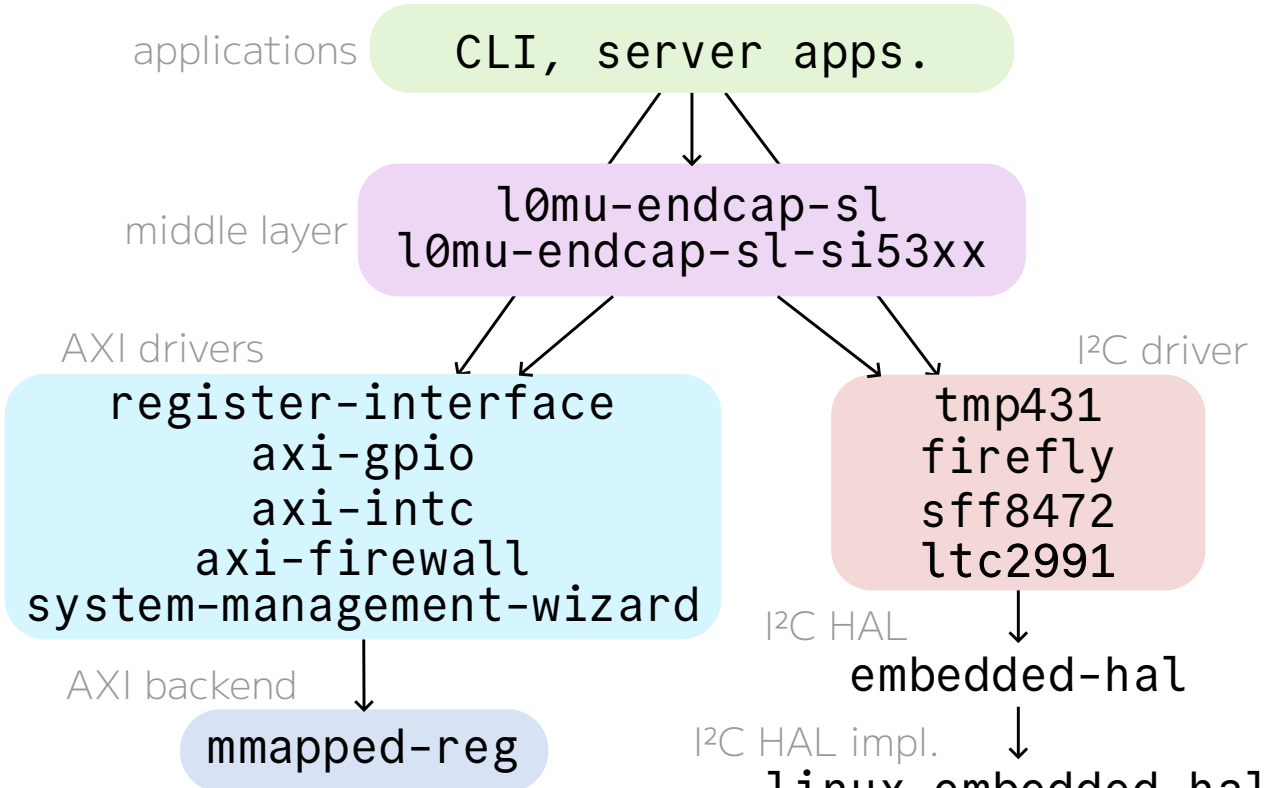
- 要件: 堅牢な開発環境、容易なビルド、安全性(安全なコード、エラーハンドリング)、ログ
- **Rust** で実装
  - 開発環境が充実
    - 言語サーバー(IDE feature via LSP)、ツールチェイン管理、ビルドシステム、パッケージ管理、ドキュメント、テスト、フォーマッタ等
  - 低レベルプログラミング、ネットワーク、CLI などに最適
  - **洗練された型システム**と標準ライブラリ: 関数型の知見
    - GC なしでのメモリ安全性、安全な並列・非同期プログラミング等
  - 普及中(OS(Win, Linux), ブラウザ, coreutils, typst, ...)
- `aarch64-unknown-linux-musl`ターゲットを使用
  - musl libc を使うことで静的にリンクされる
  - ⇒ インストールが用意
- **クロスコンパイル**も簡単になる
  - リンカをいれる
  - もしくは docker/podman ベースの cross-rs

標準で生成されるドキュメント→



ソフト構成

- **“クレート”**に分割
  - クレート: Rust におけるパッケージ単位。ライブラリもしくはバイナリ
  - コンパイル単位でもある(キャッシュ)
- 複数レイヤーに分割
- I²C ドライバーは rust-embedded の HAL に準拠
  - **Hardware Abstraction Layer(HAL)** がデバイスによらないインターフェースを提供(like I²C, SPI,)
  - I²C ドライバーは HAL にのみに依存
- AXI ドライバーも同様の設計を採用
  - `mmapped-reg`が memory mapped なインターフェース(trait)とその具体的な実装を提供
    - 提供している実装: UIO, `/dev/mem`, normal file(モック用), 他のものの一部分
  - AXI IP のドライバーは`mmapped-reg`を使用
- 上位レイヤーはこれらを用いて SL 固有のロジックを実装



目次

FPGA(VU13P)レジスタ API デザイン

- 目標
  - アドレスやレジスタの使用方法に関するミスを防ぐ
  - 5000 個以上のレジスタを階層的に整理
  - ダングリングポインタを避ける(メモリ安全性)
- Rust がネイティブでもつ言語機能を活用してこれらの目標を達成
  - 所有権、借用規則、生存期間...etc
  - **実行時の追加コストなし** — ナローポインターと同じサイズ・動的チェックなし

Rust

```rust
// レジスター/ブロックの型の例
pub struct RegA<'a> { // 'a は生存期間
    mem_ptr: *mut u32, // 生ポインタは生存期間を持たない
    // 組み込みのゼロサイズ型`PhantomData`を使用し、適切な振る舞いを与える。
    _marker: PhantomData<&'a mut Parent<'a>>,
}
```

コードとエディタにおけるフィードバック例

neovim 上で rust-analyzer(言語サーバー)によって表示される問題の例

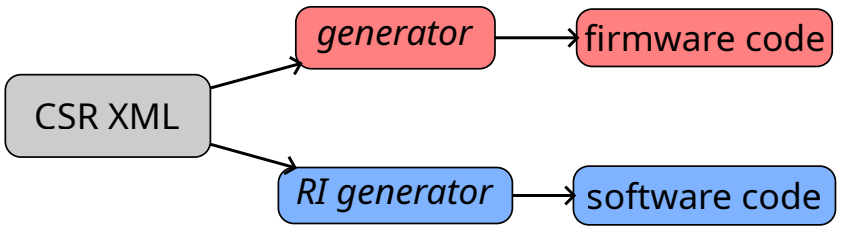
```
let ri: RegisterInterface<UioMapped> = RegisterInterface::new_with_addr()?;
    └──── E0505: binding `ri` declared here
let slr0: Slr0<'_, UioMapped> = ri.slr0();
                                └──── E0505: borrow of `ri` occurs here

// Accessing test register
let val: u32 = slr0.control().general().test()[0].read();
println!("This is u32 value: {val:#x}");
// once the parent is dropped, any children is no longer available
drop(ri);
     └──── E0505: cannot move out of `ri` because it is borrowed
           move out of `ri` occurs here
slr0.control().general().dbgled().read();
└──── E0505: borrow later used here
```

register/block object

XML でのレジスタマップと API コード生成

- FPGA レジスタ API は XML でのレジスタマップから生成
  - ファームウェアも同様に生成
- software generator
- ⇒ **このレジスタマップ自体の管理が問題に**
  - レジスタ数が膨大(> 10k LoC)
  - 分割や一部の自動生成(特に接続に関する部分)を検討中




```
csr.xml  4060
Grammars <?xml version="1.0" encoding="UTF-8"?> [1, 1]
xml <?xml version="1.0" encoding="UTF-8"?> [1, 1]
DOCTYPE:module <!DOCTYPE module SYSTEM "L0MuonSectorLogicModu
xml-model <?xml-model href="L0MuonSectorLogicModule.xsd" type
module <module name="L0MuonSectorLogicProcessor" addr="0x0000
  block <block name="SLR0"      addr="0x0000" size="0x4000" d
    block <block name="CONTROL" addr="0x0000" size="0x2000" m
      block <block name="GENERAL" addr="0x0000" size="0x100"
        register <register name="initialize_csr"  addr="0x000
        register <register name="dbgled" addr="0x0010" desc="
          field <field name="led1" mask="0x00000001" desc="DB
          field <field name="led2" mask="0x00000002" desc="DB
        register <register name="ttc_selector"  addr="0x0020"
          field <field name="enable_ttc_emu" mask="0x1" defau
        register <register name="test"  addr="0x0040" mask="0
      block <block name="GTY" addr="0x0100" size="0x100" desc
      block <block name="BCID" addr="0x0200" size="0x100" des
      block <block name="RAM" addr="0x0300" size="0x100" desc
      block <block name="DRO" addr="0x0400" size="0x100" desc
      block <block name="TRO" addr="0x0500" size="0x100" desc
```

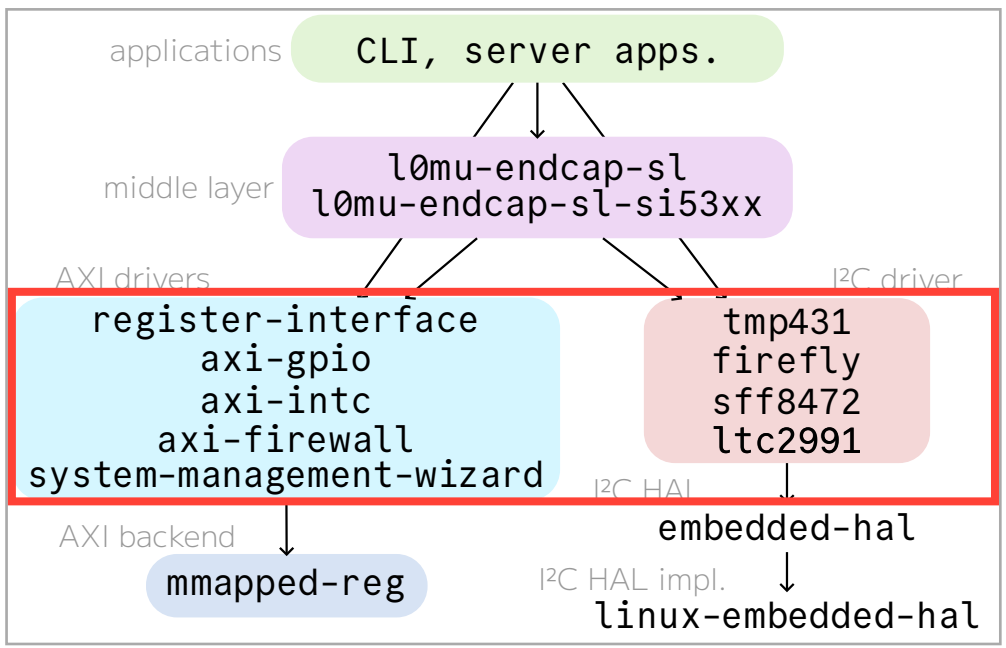
```
// PS board link ID
let link_nums: LinkNum = ri RegisterInterface<UioMapped>
    .slr0() Slr0<'_, UioMapped>
    .status() Status<'_, UioMapped>
    .psb() Psb<'_, UioMapped>
    .link_num() RegLinkNum<'_, UioMapped>
    .read();
let bank120: [bool; 4] = link_nums.bank120();
link_nums.bank
          bank120()~      Method fn(&self) -> [bool; 4]
          bank121()~      Method fn(&self) -> [bool; 4]
          bank122()~      Method fn(&self) -> [bool; 4]
```

適切な API が生成される

I²C および他の AXI IP ドライバー

- AXI IP(AXI GPIO, INTC, Sys. Man. Wiz., Firewall)と I²C 素子(TMP431, FireFly, SFP+, LTC2991)
- HAL 及び AXI バックエンドに準拠した汎用のドライバとして実装(SL 専用ではない)
- **型パラメーターを活用**
  - IP の設定や基盤の配線など静的な情報を表現
  - **型状態**(typestate)プログラミング: ステートマシンを表現(コンパイル時に検査する)

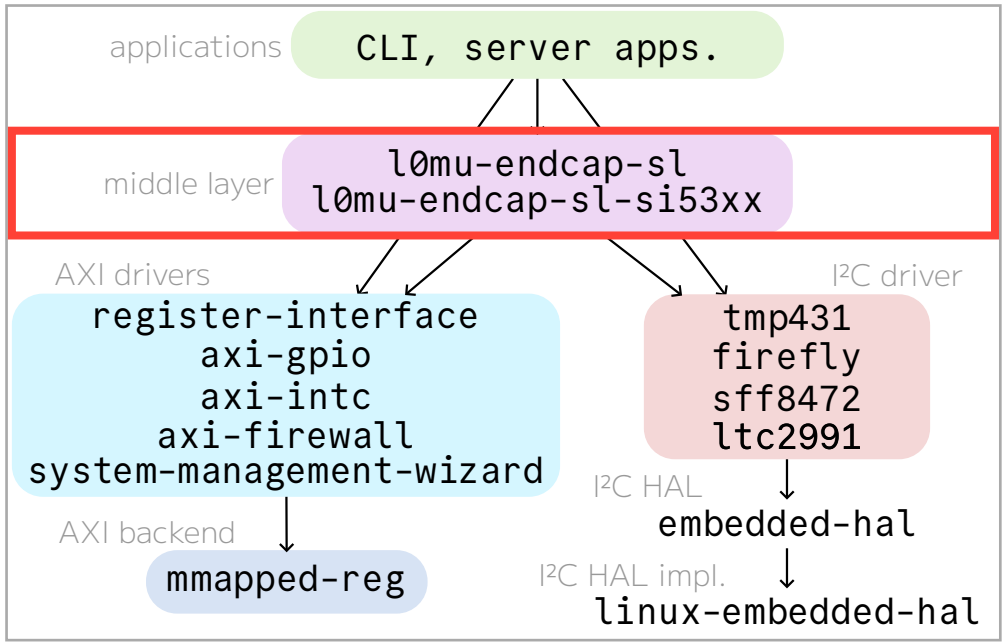
```rust
pub struct Tmp431<I2C, MODE, LOCAL, REMOTE>
where
    I2C: i2c::I2c, // HAL実装に関して多相
    MODE: marker::mode::Mode, // 型状態(変換モード)
    // 各チャンネルの状態
    LOCAL: marker::enable::Enable,
    REMOTE: marker::enable::Enable,
{
    /// 具体的なHALの実装 (e.g., I2cdev)
    i2c: I2C,
    ...
}
```



目次

中レイヤー — PS board インターフェース

- レジスタマップはファームウェア//ハードウェアの影響を強く受けている
  - 例: 1 PS board からの 3 つのリンクが別々の IO バンクに存在しうる → レジスタもばらばら
- 複数のレジスタアクセスを伴う操作がある
  - 例: PS board 上のレジスタアクセス
- PS board との通信に複数のモードが存在
- 中レイヤーはこれらを使いやすくラップする
  - リンクなどの型による分類・PS board 接続情報とのリンク(将来的に FireFly などの情報と連携)
  - 最近大幅に改修
  - TAM(やその先の JATHub)についても同様(予定)




```
let board_id: Option<PsbId> = psb &mut Psboard<Normal>
    .registers() PSboardRegisters<'_>
    .read_psboard_id(ri) Result<PsbId, PsbRegAccessError>
```

Reading a register on PS board

高レイヤー — ボード固有のロジック

- 高レイヤーでボード固有の情報を組み込んだ API を提供
- あらかじめ決まってる(AXI)アドレスや(I²C)パス(配線及び devicetree で決定)に基づいて型を定義
  — Board Support Crate(BSC)
- エラーも定義
  - Rust は`Result<T, E>`代数的データ型(ADT)による明示的なエラーハンドリングを行う(例外なし)
  - 低レイヤーで起こるエラーをラップしつつ、エラーを`enum`でまとめる
- 重要な設計のコンセプト: **CLI とサーバーで同じ型を共有する**
  - それぞれの目的に応じた trait を実装している
  - コマンド(CLI)
    - データ型: 表として表示するための`tabled::Tabled`
    - エラー型: エラースタックを表示するための`std::error::Error`
  - サーバー
    - データ型・エラー型ともに JSON へシリアライズするために`serde::Serialize`を実装
- 現在の状況: 実装をはじめた

CLI・デモサーバー実装

- アプリケーションとして CLI と HTTP server を実装
- CLI は MPSoC で直接実行。テスト・デバッグ用
- Server は将来のメイン。ウェブフレームワークの **axum** クレートを用いて実装
  - ロギングには tokio エコシステムの tracing を使用
    - 非同期プログラムに適した概念を導入 — “span”と“event”
    - 様々な“subscriber”と使用可能(標準出力や Grafana Loki, OpenTelemetry 等)
- tokio のMutexとstd::sync::Arcを使って、スレッド間でハードウェアリソースを管理
  - スレッド安産性はドキュメントを読まなくてもわかる — 型で表現されている

リクエストハンドラの例

Rust

```rust
pub(crate) async fn read_sl_temp(
    State(global_state): State<L0Mue>,
) -> Result<Json<SlTmp431Tmps>, ErrorResponse> {
    trace!("read_sl_temp called");
    let mut sl_tmp431s = global_state.sl_tmp431s.lock().expect("mutex poisoned");
    Ok(axum::Json(sl_tmp431s.read_temp()?))
}
```

CLI & Server demonstration

record

CLI & Server demonstration

CLI

record

```
wotsubo@sl-2-2-xu5-eth0-01 ~> source (./llslcli completion fish | psub )
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 -h
Si5395 commands

Usage: llslcli si5395 [OPTIONS] <SL_VERSION> <COMMAND>

Commands:
  read          Read configuration registers and check the values
  check-lol     Check LOL (loss of lock)
  write-params  Write configuration parameters
  hard-reset
  soft-reset
  help          Print this message or the help of the given subcommand(s)

Arguments:
  <SL_VERSION>  Sector logic version [possible values: v1, v2]

Options:
  -v, --verbose...  Increase logging verbosity
  -q, --quiet...    Decrease logging verbosity
  -h, --help        Print help (see more with '--help')
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 v2 hard-reset
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 v2 check-lol
u50 loss of lock
u51 loss of lock
u52 loss of lock
u53 loss of lock
Error: some Si5395 is out of lock
wotsubo@sl-2-2-xu5-eth0-01 ~ [1]> ./llslcli si5395 v2 write-params
All configuration completed
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 v2 check-lol
wotsubo@sl-2-2-xu5-eth0-01 ~>
```

CLI & Server demonstration

CLI

record

Server and client

```
wotsubo@sl-2-2-xu5-eth0-01 ~> source (./llslcli completion fish | psub )
wotsubo@sl-2-2-xu5-e
Si5395 commands

Usage: llslcli si53

Commands:
  read          Rea
  check-lol     Che
  write-params  Wri
  hard-reset
  soft-reset
  help          Pri

Arguments:
  <SL_VERSION>  Sec

Options:
  -v, --verbose...
  -q, --quiet...
  -h, --help
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 v2 hard-reset
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 v2 check-lol
u50 loss of lock
u51 loss of lock
u52 loss of lock
u53 loss of lock
Error: some Si5395 is out of lock
wotsubo@sl-2-2-xu5-eth0-01 ~ [1]> ./llslcli si5395 v2 write
All configuration completed
wotsubo@sl-2-2-xu5-eth0-01 ~> ./llslcli si5395 v2 check-lol
wotsubo@sl-2-2-xu5-eth0-01 ~>
```

```
2025-10-03T20:43:46.947019Z DEBUG mmapped_reg::uio_mmapped: found uio matching address: /sys/class/uio/uio2, addr=0x80040000, map=/sys/class/uio/u
io2/maps/map0
2025-10-03T20:43:46.947198Z  INFO l0mue_server: global state constructed
2025-10-03T20:43:46.947829Z  INFO l0mue_server: router constructed
2025-10-03T20:43:46.947952Z  INFO l0mue_server: serving the server...
2025-10-03T20:43:50.317747Z TRACE axum::serve: connection 130.87.28.60:49500 accepted
2025-10-03T20:43:50.318377Z DEBUG request{method=GET uri=/demo/read_deadbeef version=HTTP/1.1}: tower_http::trace::on_request: started processing
request
2025-10-03T20:43:50.318451Z TRACE request{method=GET uri=/demo/read_deadbeef version=HTTP/1.1}: l0mue_server: from extractor: /demo/read_deadbeef
2025-10-03T20:43:50.318617Z TRACE request{method=GET uri=/demo/read_deadbeef version=HTTP/1.1}: mmapped_reg::uio_mmapped: reading /sys/class/uio/u
io1/maps/map0
2025-10-03T20:43:50.318712Z TRACE request{method=GET uri=/demo/read_deadbeef version=HTTP/1.1}: mmapped_reg::uio_mmapped: /sys/class/uio/uio1/maps
/map0/addr: 0x00000000a0000000

2025-10-03T20:43:50.318743Z DEBUG request{method=GET uri=/demo/read_deadbeef version=HTTP/1.1}: mmapped_reg::uio_mmapped: found uio matching addre
ss: /sys/class/uio/uio1, addr=0xa0000000, map=/sys/class/uio/uio1/maps/map0
2025-10-03T20:43:50.318919Z DEBUG request{method=GET uri=/demo/read_deadbeef version=HTTP/1.1}: tower_http::trace::on_response: finished processin
g request latency=0 ms status=200
^C2025-10-03T20:45:09.859210Z TRACE axum::serve: received graceful shutdown signal. Telling tasks to shutdown
2025-10-03T20:45:09.859339Z TRACE axum::serve: signal received, not accepting new connections
2025-10-03T20:45:09.859469Z TRACE axum::serve: waiting for 0 task(s) to finish
2025-10-03T20:45:09.859757Z  INFO l0mue_server: exitting the app
wotsubo@sl-2-2-xu5-eth0-01 ~>
```

```
~ ⎈ 10GiB/13GiB | 19GiB/40GiB
 77% at 05:43:44 fsh → curl -i -X GET '130.87.240.59:3000/tdaq/demo/read_deadbeef'
HTTP/1.1 200 OK
content-type: application/json
content-length: 10
date: Fri, 03 Oct 2025 20:43:50 GMT

3735928559↵
~ ⎈ 10GiB/13GiB | 19GiB/40GiB
 77% at 05:43:50 fsh → julia --sysimage $JL_SYSIMG_ETC --project -t 12 -E "3735928559 |> UInt"
0x00000000deadbeef
~ ⎈ 10GiB/13GiB | 19GiB/40GiB
 77% at 05:44:09 fsh →
```

目次

レジスタマップのフォーマット設計・移行計画

- レジスタマップからの自動生成を始めた経緯
  - 組み込み Rust のエコシステムを調べていると svd2rust というものがあった
    - マイコンのレジスタマップのフォーマット SVD から自動でコードを生成
  - 更に調べると、ATLAS の他グループで XML のレジスタマップからのコード生成をしていた
- 去年 12 月頃から XML のデザインなどを開始
- 本番運用に向けたレジスタの整理に追加する形で始まった
  - 当初はレジスタ整理→XML 化としていたのが現在は同時にやることになった
- **移行に時間がかかっている**
  - 原因 1: レジスタマップが非互換なため、ソフトと合わせたアップデートが必要
  - 原因 2: ファームウェアでより重大なバグが度々見つかり、その修正が優先される
    - ファームウェア開発は遅い(ビルド時間・従事する人員多)
- **ソフトウェアの実機検証も合わせて遅れた**
  - 現在は新旧レジスタマップの差分をレジスタ API・中レイヤーで吸収
    - 新レジスタマップでしか使えない最適化などは(当初は入っていたが)除いている
- CPU + FPGA でかつ大量のリンクがあるようなシステムに最適なレジスタマップの形式を模索中

大規模なリファクタリング

- これまでの開発で大幅なデザインの変更を複数回してきた
  - レジスタ API は 3 つのプロトタイプがあった
    - 書きながら新しいアイデアがでて比較検討して決定
  - AXI 系のバックエンド`mmapped-reg`ははじめレジスタ API と分離していなかった
    - テストできるようにしたいといったことや、AXI GPIO などでも使い回せるとあとで気づいて分離した
  - PS board API は他のボードの API や FireFly 光モジュールの制御との統合を考えて「リンク」の概念を中心としたものに書き換え
    - モジュール構造の見直しなどによって旧レジスタマップとの互換性も上がった
- このように**開発を進めるにつれて理解が深まり、既存のコードを度々大改修した**
- Rust はこのようなリファクタリングがスムーズだった
  - 静的な検査によって安心感がある(“empowered”)
  - 言語サーバー・コンパイラが親切(エラーメッセージがわかりやすい、ヒントを教えてくれる)
  - モジュールが構造化を助けてくれる

割り込み

- ここまでのものはすべてソフトウェア側が主体
- 一方、SL や MPSoC、AXI IP には**割り込み**のための仕組みが用意されている
  - SL: I²C 素子の status 線などを MPSoC PL に配線
  - MPSoC: PL から PS への割り込み線(16bit)
  - AXI IP: 大抵 irq 線がある。
    - AXI Chip2Chip には割り込みの通信機能もついている(4bit)
  - PS board-SL 間の通信では40 MHzで主要な部品の状態を垂れ流している
- これらを活用したい
- ソフトウェア側での受け方
  - devicetree で MPSoC 内の GIC(Global Interrupt Controller)の ID を UIO に割り当てられる
  - UIO では POSIX システムコール`read`で割り込みを読める(blocking)(`read(uio_fd, ...)`)
- この原理は検証済み
  - 温度センサーや AXI IP で閾値を変えて割り込みの発生を確認
- 単純に割り込みを無限ループで回すと CPU リソースが無駄になる・プログラムを理解しづらくなる
  - Rust の非同期インターフェースで扱いたい
- 今後: Linux の`epoll`が使えるはずなので、`tokio`のシステムに組み込む

```
&axi_intc_0 {                                    dts
    compatible = "generic-uio";
    interrupt-parent = <&gic>;
    interrupts = <0 92 4>;
};
```

目次

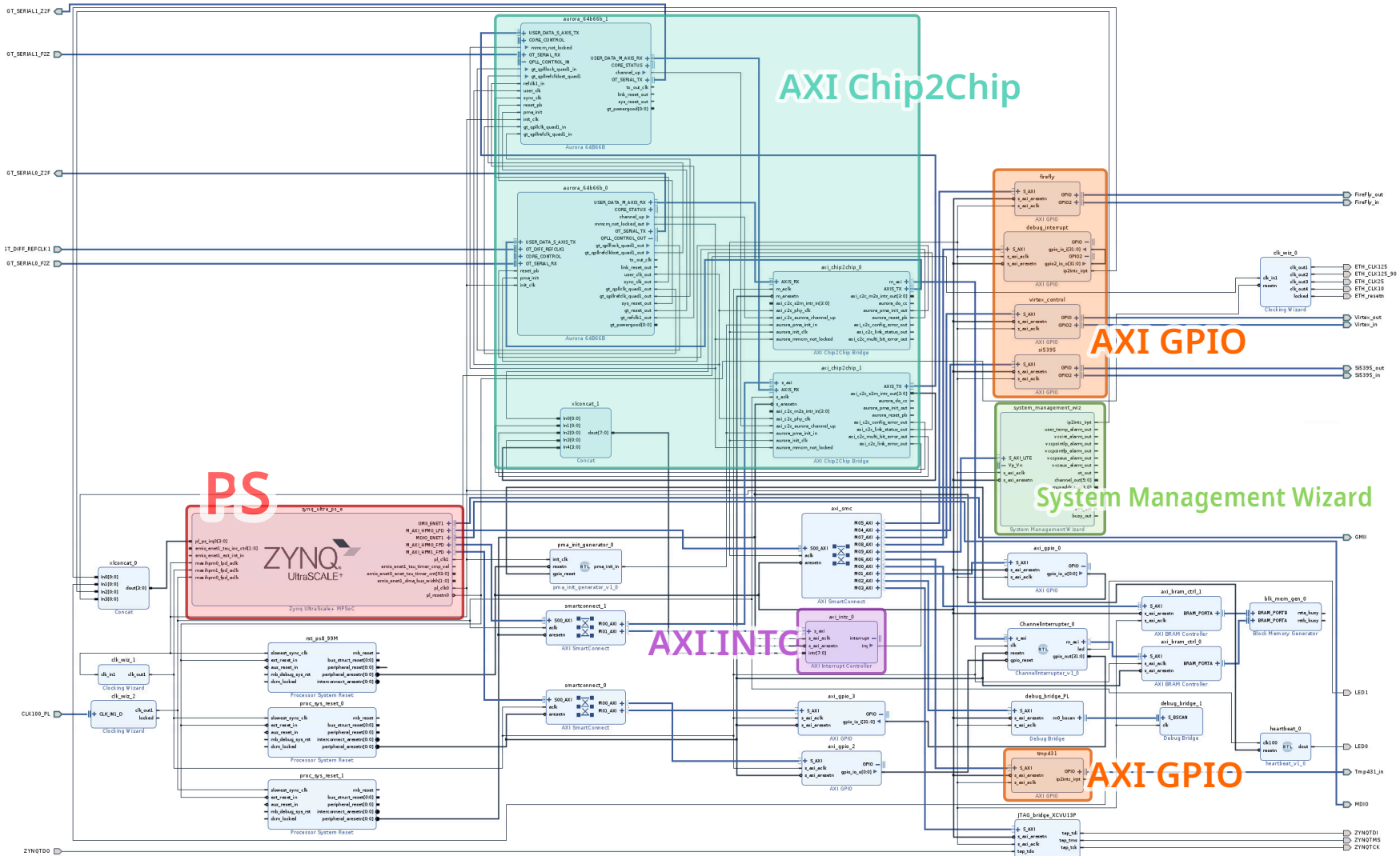
まとめ

- ATLAS 初段エンドキャップミューオントリガーでは、**Zynq UltraScale+ MPSoC** と **Virtex UltraScale+ FPGA** を搭載したボードを使用
- MPSoC で Linux とサーバーをユーザースペースで実行
  - 外部 PC 上のアプリを介して ATLAS と統合
- アプリケーションは **Rust** で実装
- 低レイヤー: I²C ドライバー、AXI IP ドライバー、FPGA レジスタ
  - unsafe な libc や生ポインタを“Rusty”なインターフェースで wrap
  - 型システムを活用
- 高レイヤー: フロントエンドボード API やボード固有のデバイス・エラー定義
- アプリケーション: CLI とサーバー
- レジスタマップの移行・レジスタマップ管理方法・割り込みの処理などに課題あり

付録

PL firmware block design

- AXI GPIO handles external digital I/O
- AXI INTC handles interrupts
  - currently used for interrupts over/from AXI Chip2Chip
- AXI Firewall will be added in front of AXI Chip2Chip



Register map XML development environment

- Register map XML uses **XML schema**(XSD file) to validate the XML
  - Test basic structure, attribute requirement
  - Checks additional naming rules, structure rules using XPath
  - Using `xmllint --validate` from shell, `lemminx` while editing(completion, diagnostics via LSP)
- Software code generator also implements additional validations
  - Address duplication detection
  - Register address is within the range
  - Field mask duplication detection
- Using **Nix** to integrate these
  - “Nix is a purely functional package manager” (Nix 2.28 reference manual)
  - Just `nix run .#test` to do all
    - Automatically fetches all required resources/programs including costom code generator then builds them and run
    - All dependencies are recorded in `flake.lock` file
  - Also used in CI — ensured reproducibility