PULPプロジェクトのRISC-Vコア向けインターフェース

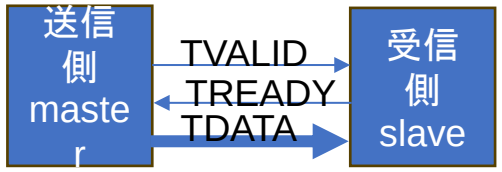
2025.2.27 RISC-V day Tokyo Spring

天野英晴、小島拓也（東大）
久我守弘（熊本大）
奥原颯（シンガポール国立大）
飯田全広（熊本大）

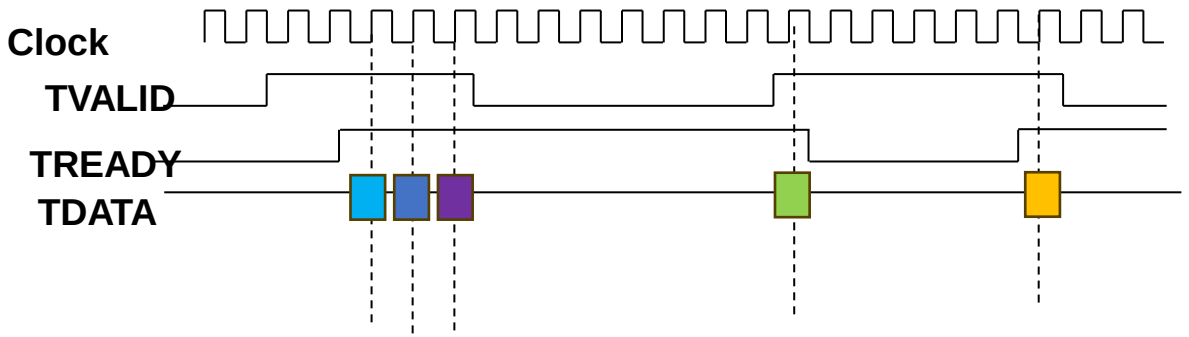
RISC-V Coreの組み込み

- RISC-Vの大きな利点は様々なオープンソースCoreが公開されていること
- メモリ、周辺I/Oを接続するインタフェースが必要
  - 標準バスを用いて接続

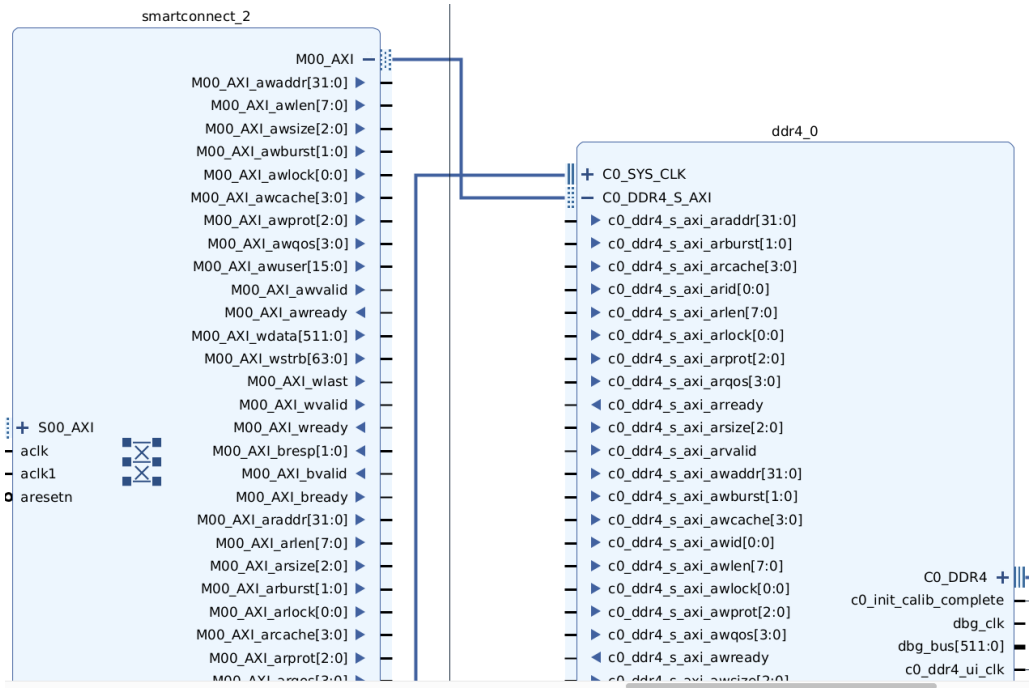
  → AXIバス、AMBAバスなど
- 複雑、多くの場合オーバースペック
- 拡張性に欠く
- FPGAでは標準環境があるが、一般的には存在しない



TVALD=1, TREADY=1の時
TDATAが転送される



AXI Stream



AXI Bus

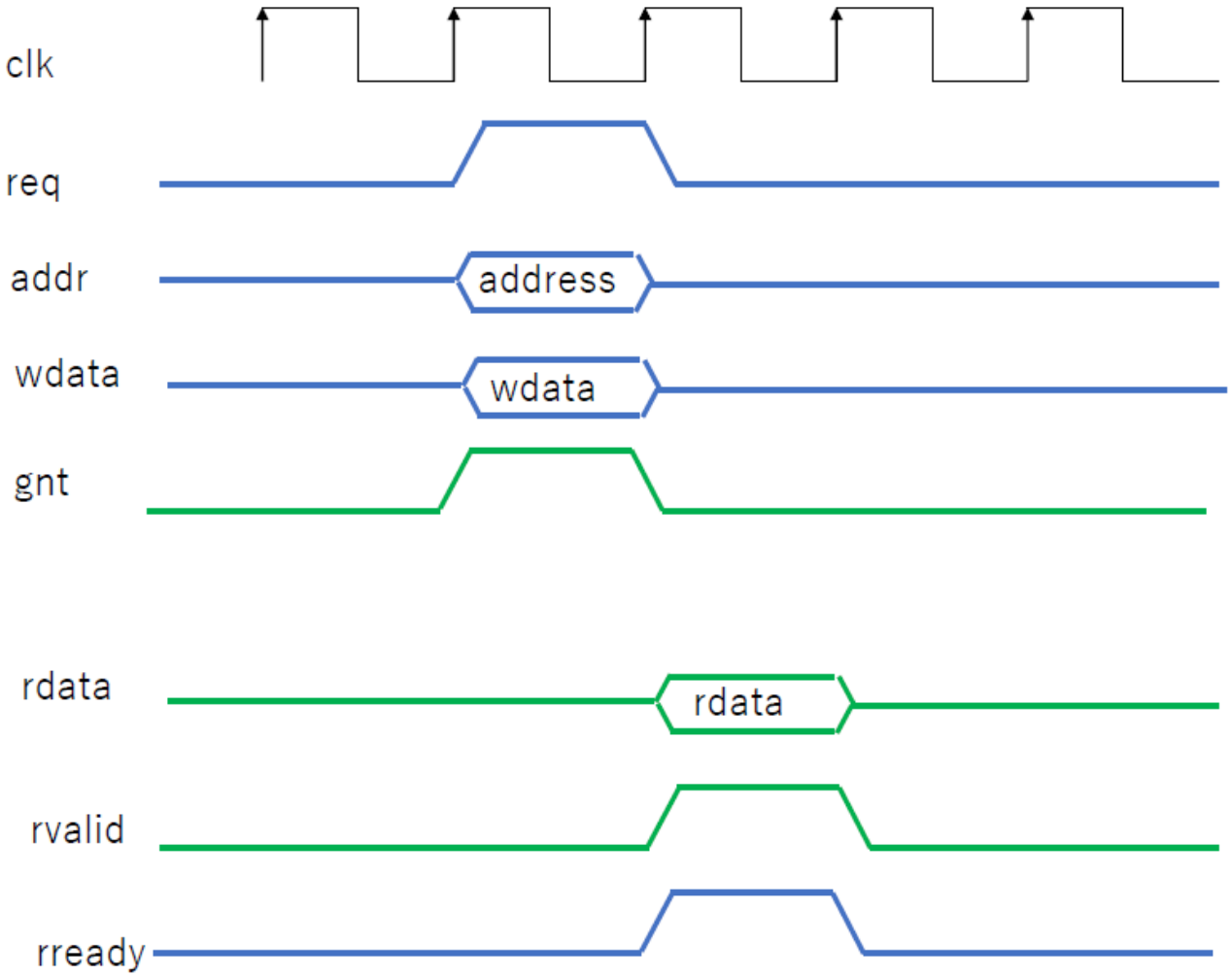
PULP(Parallel Ultra-Low-Power) Project

- ETH Zurich ／Univ. of Bologna のProf. Luca Beniniらのプロジェクト
https://github.com/pulp-platform/pulp

- RISC-Vに関連する様々なオープンソースIPを公開
- 標準インタフェースOBIでIP間を接続
- System Verilogの機能を用いたスマートな記述

- このIPを用いて作成したSLMLET-IIを例にOBIの利用手法を解説

標準IO　OBI（Open Bus Interface）



Manager：　青色
Subordinate:　緑色

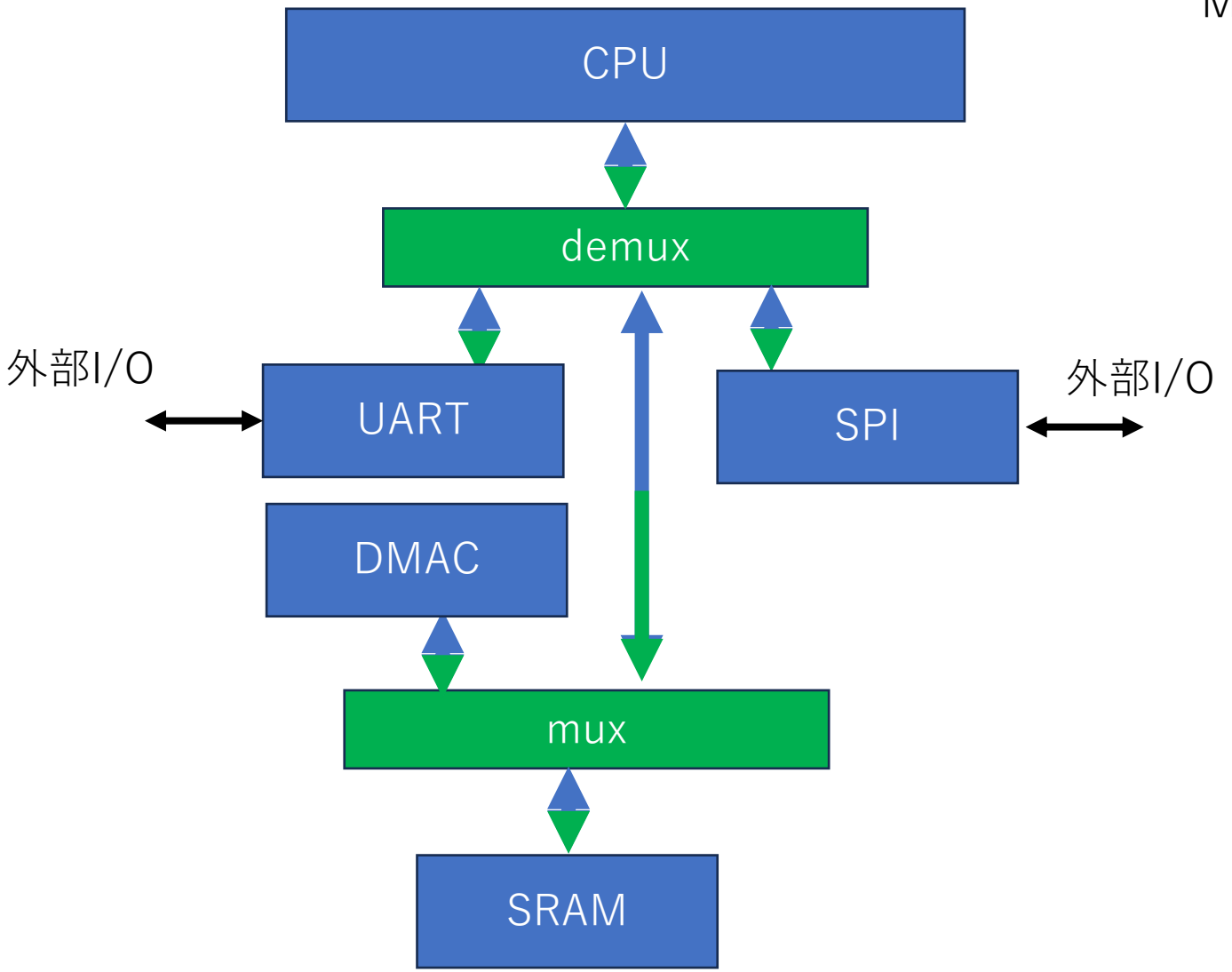
単純なインタフェース
System VerilogのInterface文で定義される

```
interface OBI_BUS #(
  parameter obi_pkg::obi_cfg_t OBI_CFG          = obi_pkg::
       ObiDefaultConfig,
  parameter type               obi_a_optional_t = logic,
  parameter type               obi_r_optional_t = logic
) ();
  logic                              req;
  logic                              reqpar;
  logic                              gnt;
  logic                              gntpar;
  logic [  OBI_CFG.AddrWidth-1:0]    addr;
  ...
```

cv32e40s他、各種IPが準備されている

System Verilogの高レベル記述を利用

インタフェース構築の考え方



Manager　　Subordinate

OBIを持ったmux, demux, switchによりネットワークを自在に構成

System Verilogの機能で個別化できる

記述例: demux

```
module obi_demux #(
/// The OBI configuration for all ports.
parameter obi_pkg::obi_cfg_t ObiCfg      = obi_pkg::ObiDefaultConfig,
 /// The request struct for all ports.
parameter type               obi_req_t   = logic,
/// The response struct for all ports.
parameter type               obi_rsp_t   = logic,
/// The number of manager ports.
parameter int unsigned       NumMgrPorts = 32'd0,
 …)
(  input  logic                  clk_i,
 input  logic                  rst_ni,
 input  select_t                sbr_port_select_i,
 input  obi_req_t               sbr_port_req_i,
 output obi_rsp_t                 sbr_port_rsp_o,
output obi_req_t [NumMgrPorts-1:0] mgr_ports_req_o,
input  obi_rsp_t [NumMgrPorts-1:0] mgr_ports_rsp_i);
```

まずパラメータを設定

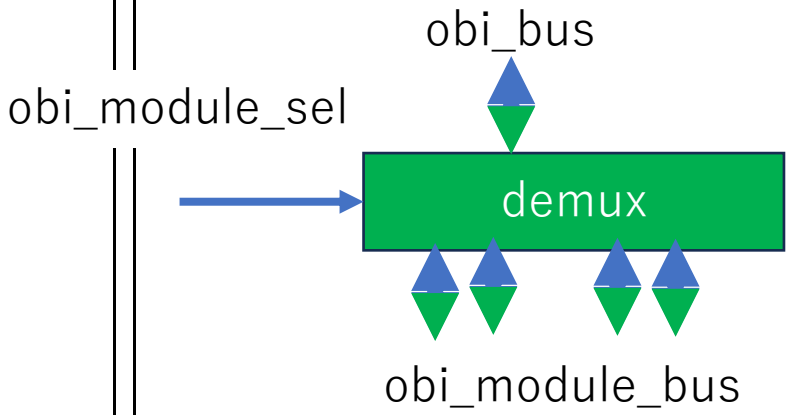
本当はSubordinateとManagerを分けて書きたいのだが、Verilog文法では入出力を分けて書く必要がある

Subordinateのポート

Managerのポート

demuxの呼び出し

```
assign obi_module_sel = (is_range(SHARED_MEM_RANGE,
                  obi_bus.addr)) ? 2'b00 :
        (is_range(OBI_REG_RANGE, obi_bus.addr)) ? 2'b01:
        (is_range(OBI_FPGA_RANGE, obi_bus.addr)) ? 2'b10:2'
              b11 ;
    obi_demux_intf #(
        .ObiCfg(slmlet_pkg::ObiCfg),
        .NumMgrPorts(4),
        .NumMaxTrans(2)
    ) demux_obi_periph (
        .clk_i(sys_clk),
        .rst_ni(reset_n),
        .sbr_port_select_i(obi_module_sel),
        .sbr_port(obi_bus.Subordinate),
        .mgr_ports(obi_modules_bus.Manager)
    );
```



wrapperの記述

上位モジュール記述

```
scanc_intf #(
    .ObiCfg(slmlet_pkg::ObiCfg)
) scanc_intf0 (
  .obi_mgr(obi_shared_mem_sbrs[3].Manager),
   .clk(sys_clk),
 .rst_n(reset_n),
...
);
```

面倒だがこれを
やらないとCADが
受け付けない

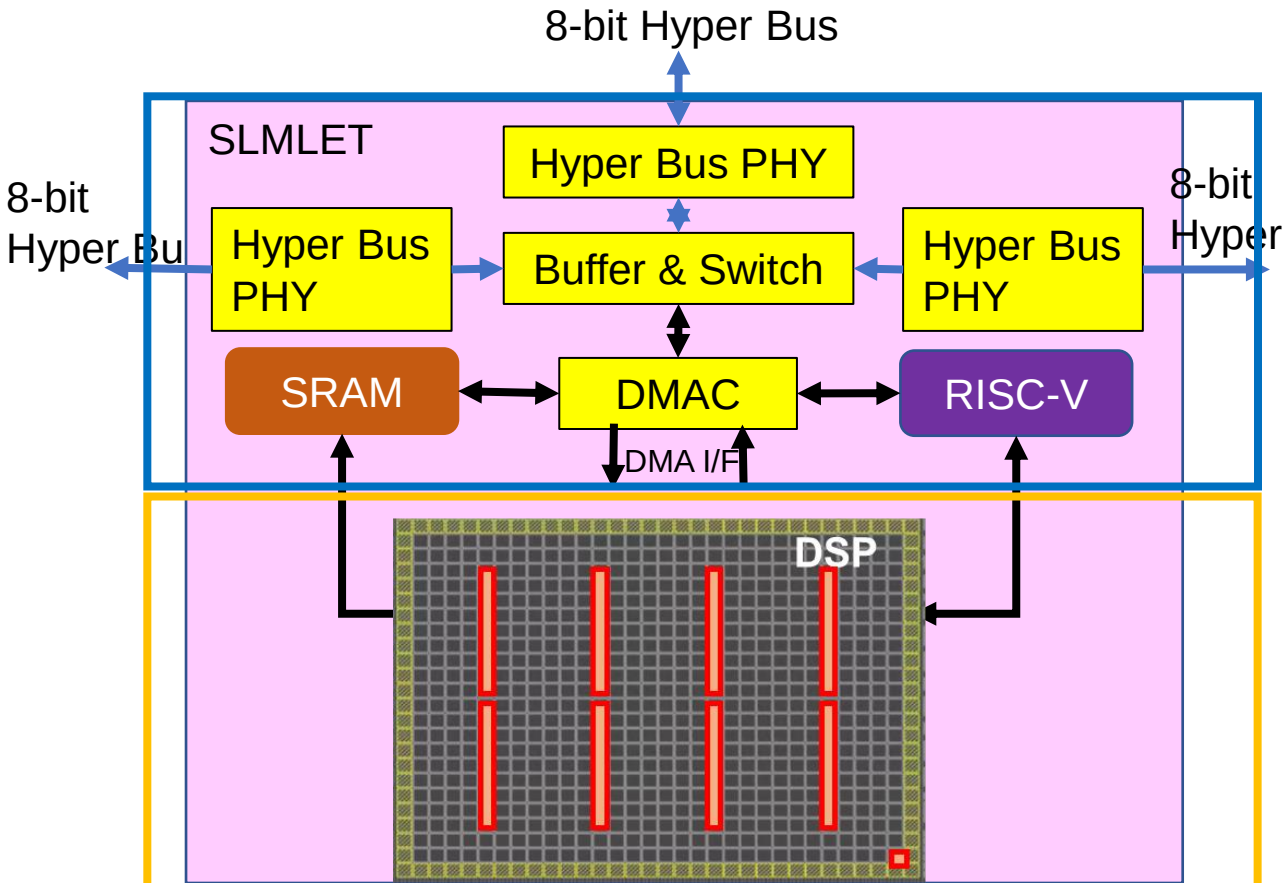
本体の呼
び出し

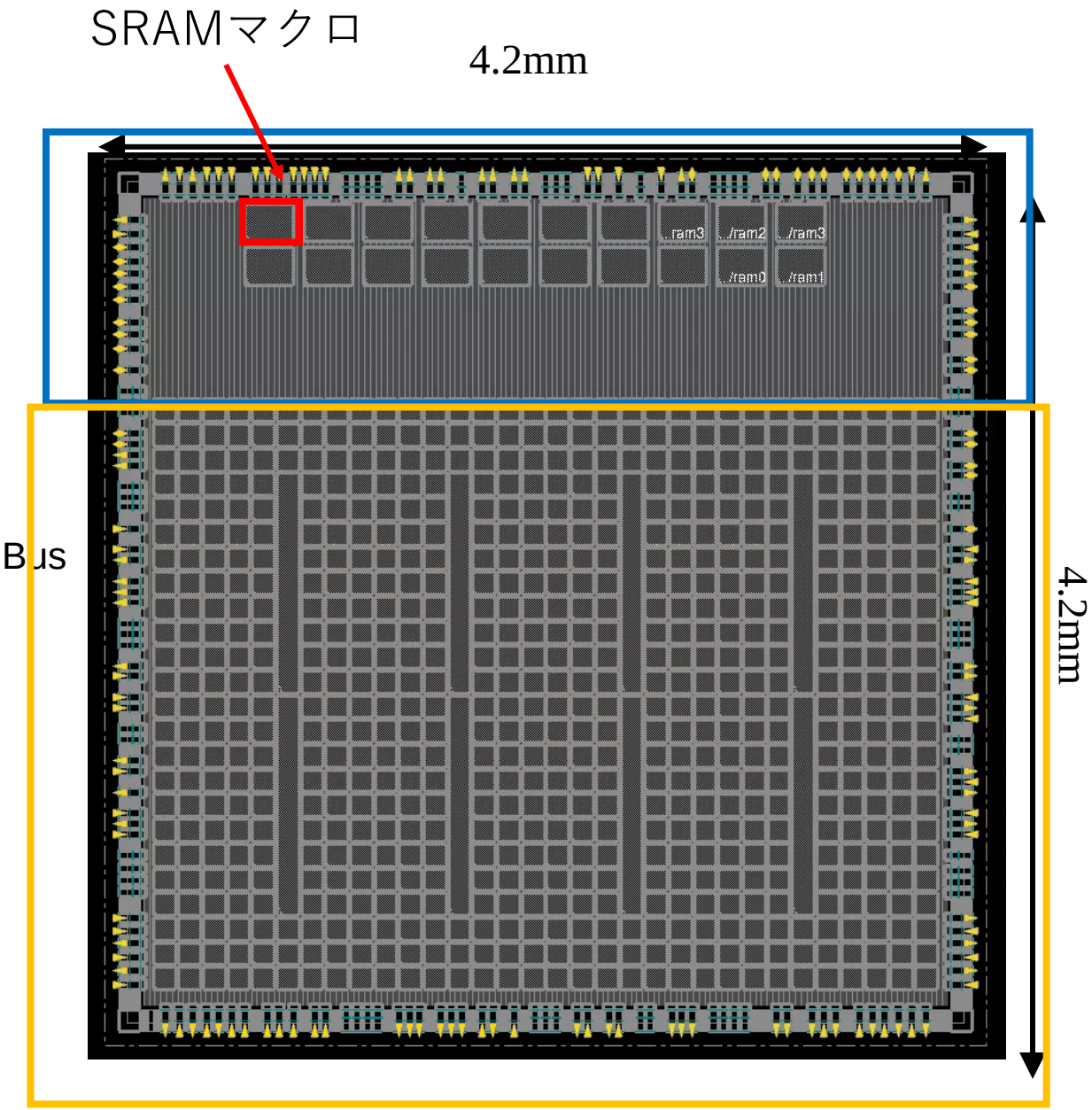
wrapper

```
module scanc_intf import obi_pkg::*, slmlet_pkg::*; #(
  parameter obi_pkg::obi_cfg_t ObiCfg = obi_pkg::
        ObiDefaultConfig
) (
 input clk, rst_n,
 OBI_BUS.Manager obi_mgr,
 input [1:0] scanreq,
 ...
) ;
  `OBI_TYPEDEF_ALL(mgr_port_obi, ObiCfg)
  mgr_port_obi_req_t mgr_ports_req;
  mgr_port_obi_rsp_t mgr_ports_rsp;
 `OBI_ASSIGN_FROM_REQ(obi_mgr, mgr_ports_req, ObiCfg)
 `OBI_ASSIGN_TO_RSP(mgr_ports_rsp, obi_mgr, ObiCfg)

scanc #(
    .ObiCfg           (ObiCfg),
    .mgr_port_obi_req_t ( mgr_port_obi_req_t    ),
    .mgr_port_a_chan_t  ( mgr_port_obi_a_chan_t ),
    .mgr_port_obi_rsp_t ( mgr_port_obi_rsp_t    ),
    .mgr_port_r_chan_t  ( mgr_port_obi_r_chan_t )
) scanc_0(
 .clk(clk),
 .rst_n(rst_n),
...
 // Manager For SRAM
.mgr_port_obi_req_o(mgr_ports_req),
.mgr_port_obi_rsp_i(mgr_ports_rsp)
);
endmodule
```

IOBで作ったSoCの例：SLMLET-2　(SoC型FPGA)

- 4.2mm□ USJC 55nm
- 面積の70％をFPGA部が占める

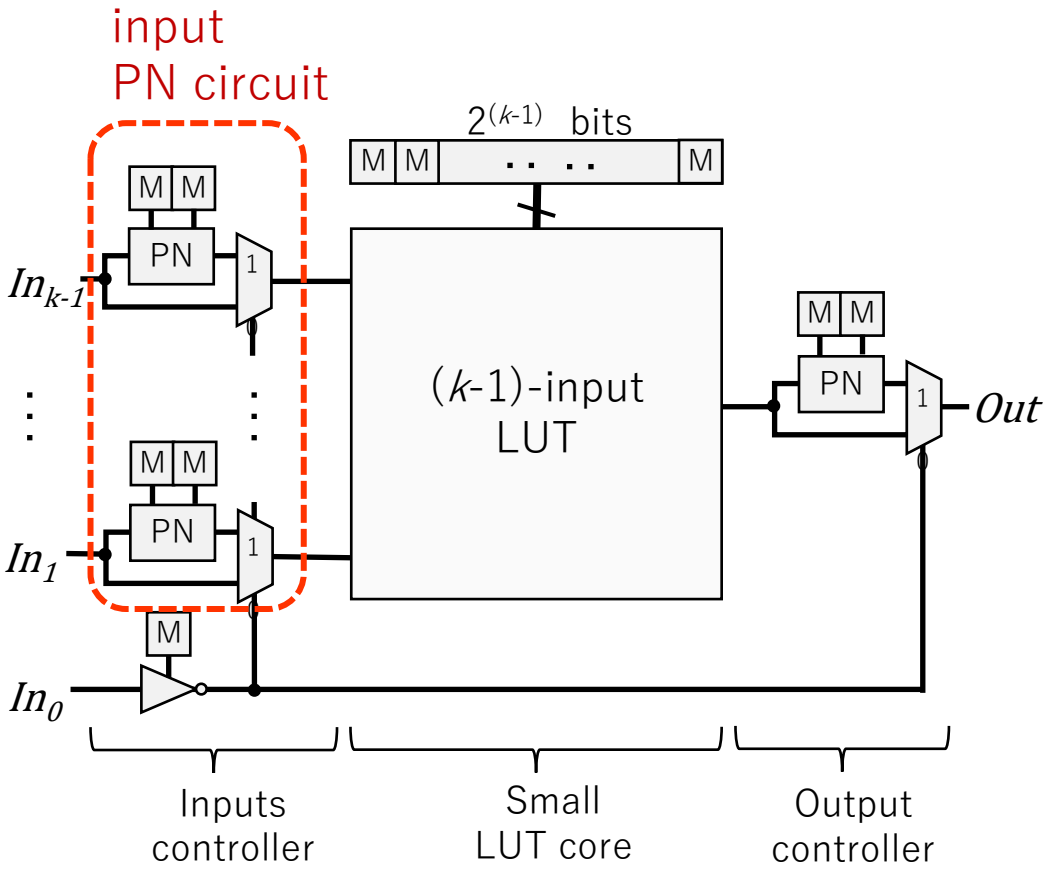




SLM(Scalable Logic Module)とは？

熊本大で開発中のFPGA [FPL2014]

- シャノン展開を使うことで Configuration Memoryの必要量を減らすことができる
  （NPN representation）
- k-input SLM エレメントは O(2^{k-1}).
- 入力PN circuitの構成により実現できるロジックと、Configuration Memoryの要求量とのトレードオフを変えることができる.



k-int SLM cell

RISC-V core

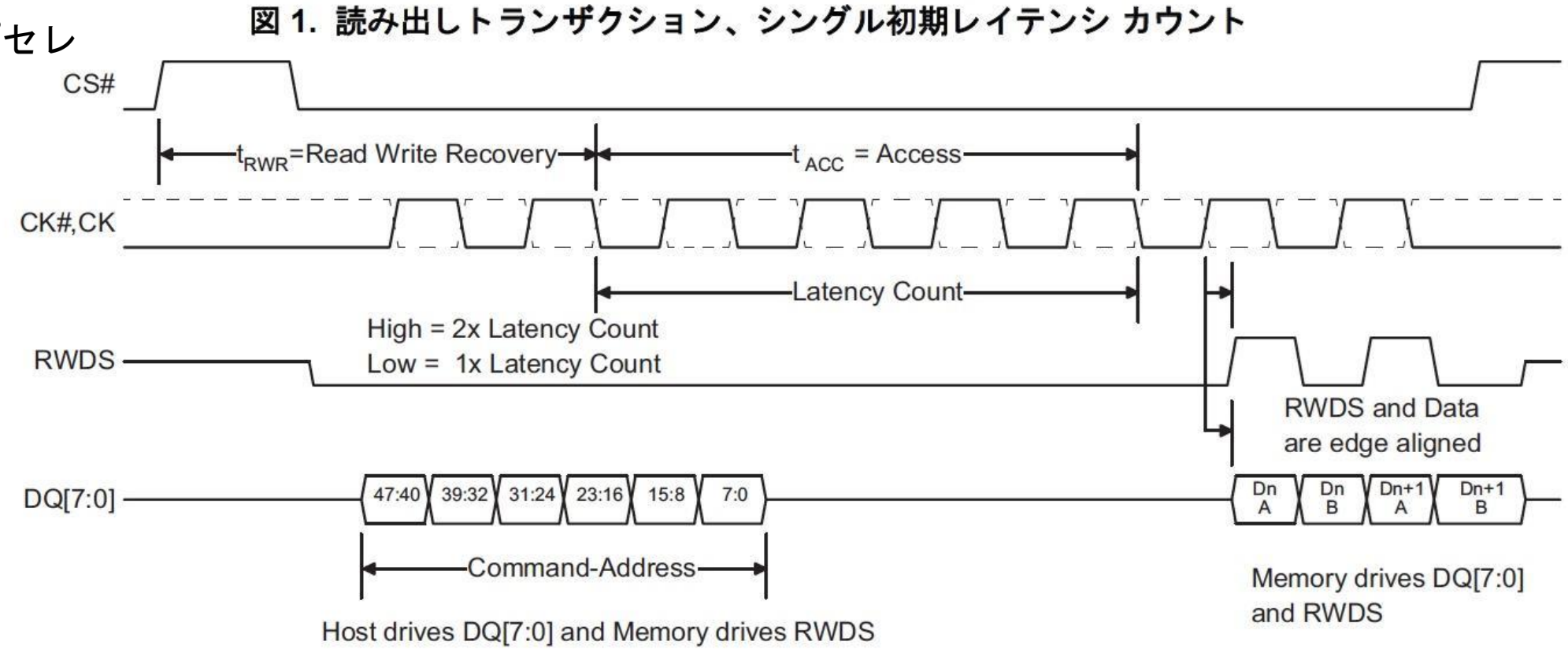
- CV32E40S、5-stage pipeline，instruction/data cache.
- 教科書的なパイプライン構造を持つ
  - cv32e40s_if_stage.sv
  - cv32e40s_id_stage.sv
  - cv32e40s_ex_stage.sv
  - cv32e40s_wb_stage.sv
- Standard I/O OBIを利用しているが、周辺はAXI busを使う
- DRAMインタフェースと外部ネットワークにはHyperbus
- キャッシュのインタフェースは自家製のOBIもどきを使った

入出力バスHyperBus/HyperRAM?

- HyperBus
  - SPIの高性能版、xSPIとも呼ばれる
  - 通常のチップI/Oでも動作する
  - そこそこの性能(400MB/s:3.2Gbps). USB2.0、1Gb Ethernetと同程度.
  - HyperRAMが直結可能.
  - Free IPs が利用できる
  - 問題点：まだあまり流行っていない
- HyperRAM
  - 組み込み用DRAM、そもそもDDR4/5はEdge Computingにはオーバースペック.
  - 128Mbit.
  - 超低消費電力: 35uW sleep power.
  - HyperBusに直結.

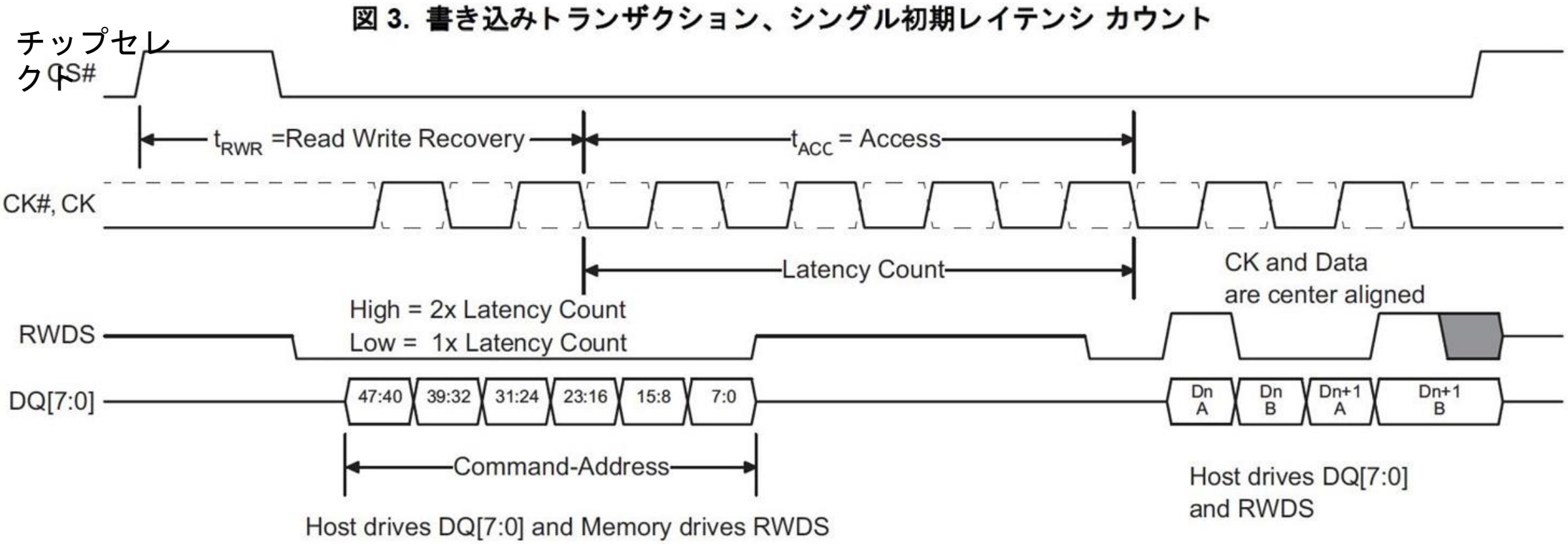
Hyperbus protocol (Read)

チップセレクト



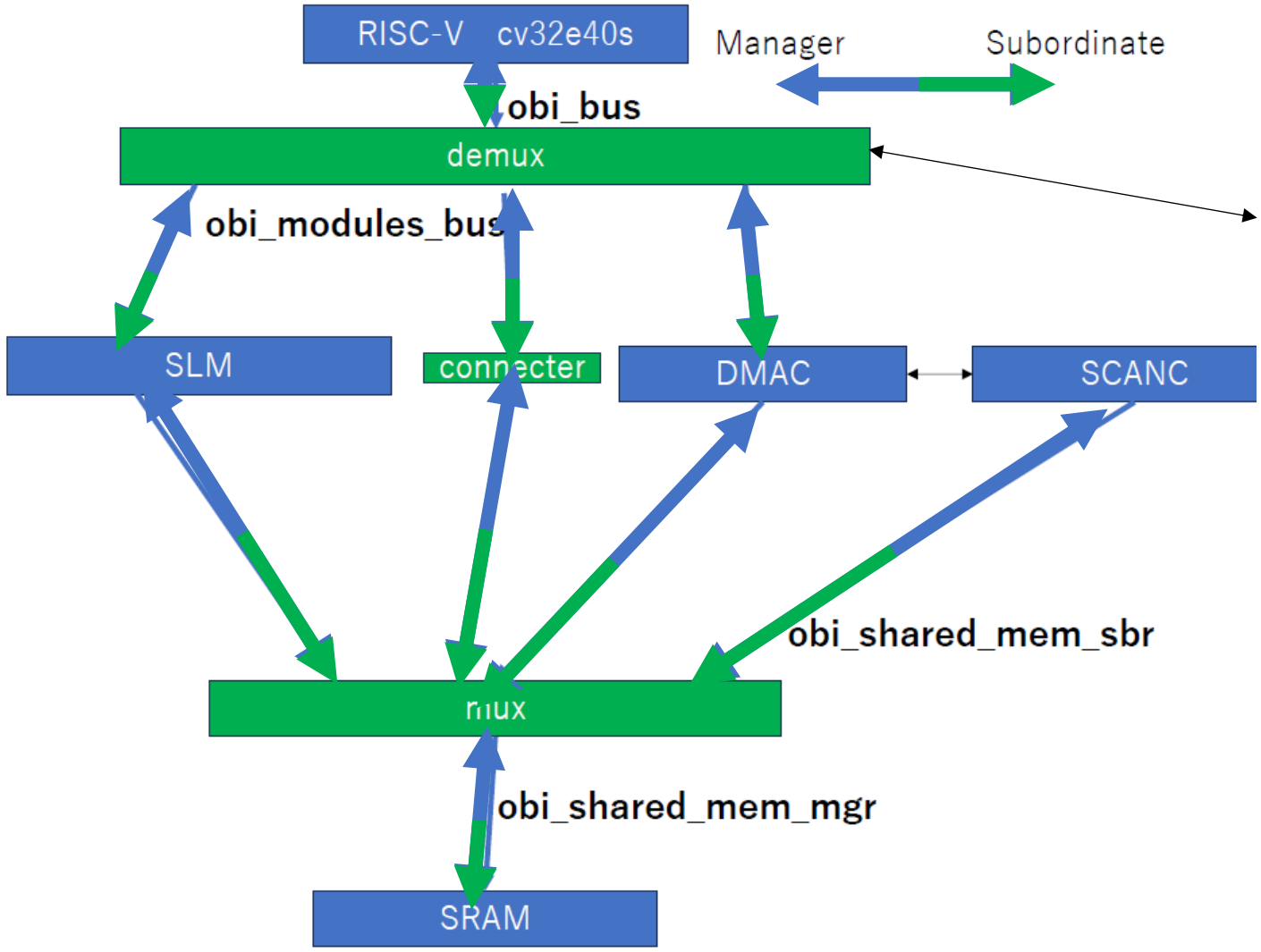
RWDSはプロトコル前半はLatency countが2倍か１倍かのチェックに使われ、
後半では入力データDQに同期したクロックになる

Hyperbus protocol (Write)



RWDSはプロトコル前半はLatency countが2倍か１倍かのチェックに使われ、
後半では出力データDQのデータマスクになる

標準IPを用いてモジュールを接続できる




```
assign obi_module_sel = (is_range(SHARED_MEM_RANGE,
                  obi_bus.addr)) ? 2'b00 :
       (is_range(OBI_REG_RANGE, obi_bus.addr)) ? 2'b01:
       (is_range(OBI_FPGA_RANGE, obi_bus.addr)) ? 2'b10:2'
            b11 ;
   obi_demux_intf #(
       .ObiCfg(slmlet_pkg::ObiCfg),
       .NumMgrPorts(4),
       .NumMaxTrans(2)
   ) demux_obi_periph (
       .clk_i(sys_clk),
       .rst_ni(reset_n),
       .sbr_port_select_i(obi_module_sel),
       .sbr_port(obi_bus.Subordinate),
       .mgr_ports(obi_modules_bus.Manager)
   );
```

OBIを用いたIP設計は世界を救うか？

- 現在の記述法は、柔軟性と合成可能性を両立させたかったため
  - intfモジュールでwrapしないと合成できない
  - Configを切り替えることで様々な派生インタフェースが使える
  - 少なくとも上位階層ではうまく繋げる
  - シミュレーションの可視性も優れている
- しかし、以下の欠点がある
  - 記述量が多い
  - 記述が分散していて可読性が低い
  - 合成はDesign CompilerのV-2023.12-SP2では可能だったが、古い版ではダメ、Xilinx社のFPGAではダメ
  - シミュレータはncverilogはダメ　xcellium,verilaterはOK

SLMLET-2では理解するまでは時間が掛かったがデバッグは確かに楽だった

Edge用に適したアーキテクチャは？

- 低コストで低消費電力
- 特定のアプリケーションには高性能
- 通信機能が必要、ネットワーク、スイッチ内蔵
- 柔軟性と拡張性を兼ね備える
  - マルチチップ構成が簡単に実現できる
- チップの設計、製造が簡単
  - 様々な用途のチップを簡単に作れる

→　Shell-Role type architecture

- 最初はFPGAでこのアーキテクチャを実現してきた

→　FiC/M-KUBOS clusters

Partial reconfiguration でShellとRole部を分離していた

- FPGAでは高コスト
  - SLMLET/SLMLET-2
  - Agile-X PF (東大dlab)

Shell-Role Type Chip: A universal device for IoT

Shell

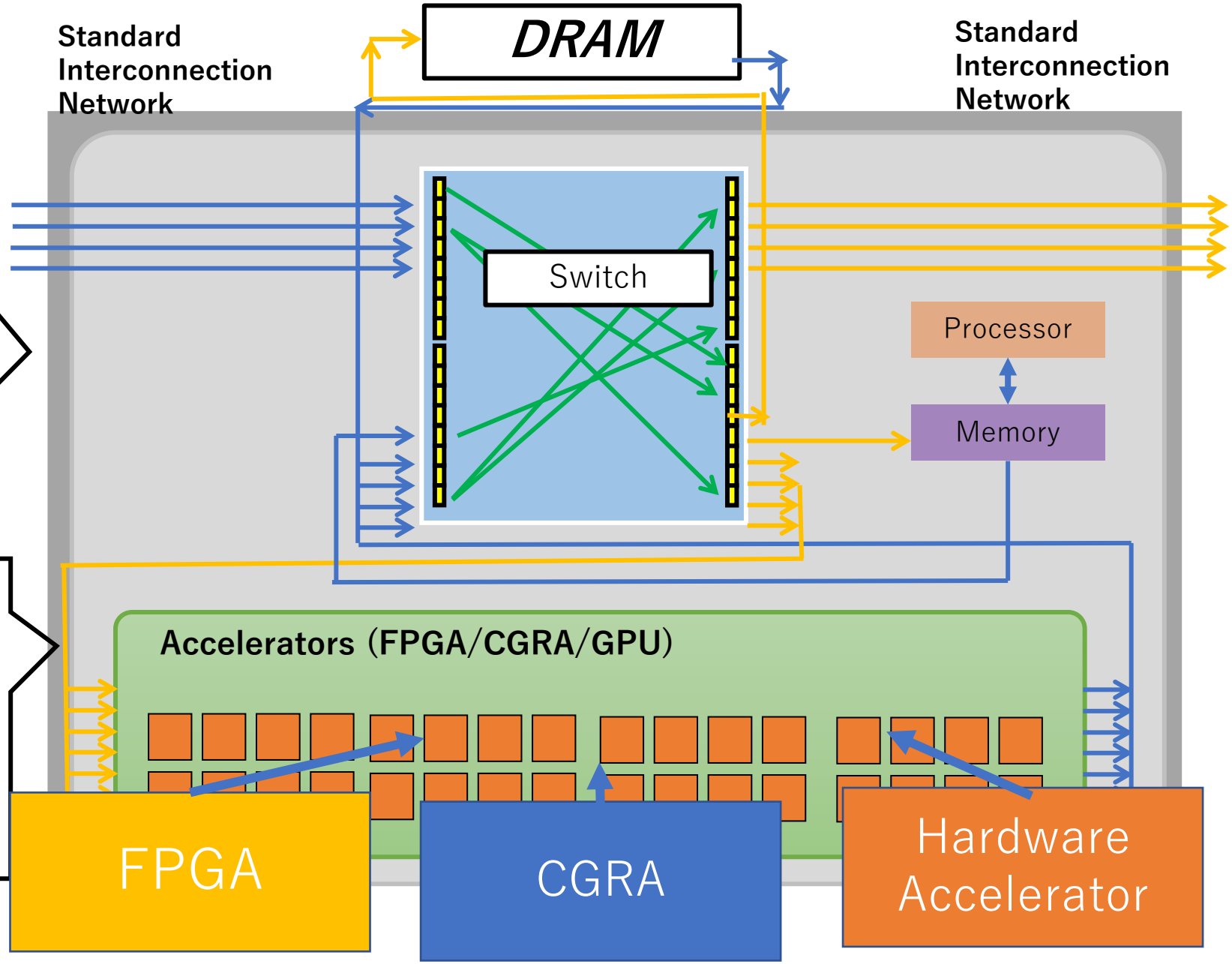
Fixed Region→ The layout is fixed

- Standard Interconnection Network
- Switching Function
- Processor
- On-Chip Memory

Role

Switchable Region→ Depending on the application

- FPGA
- CGRA
- GPU
- Dedicated Hardwired Logic



SLMLET：Shell-RoleアーキテクチャのProof of Concept

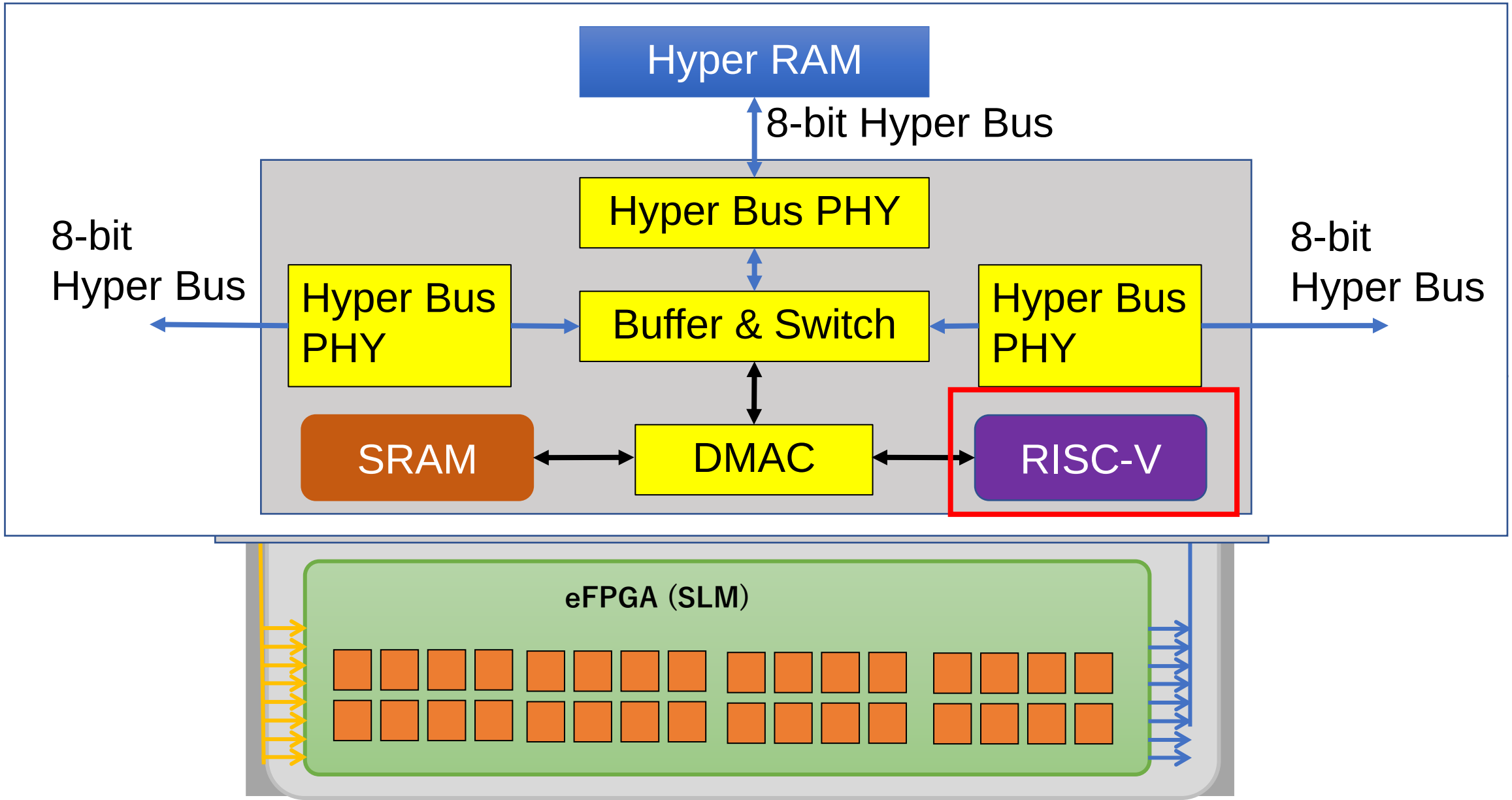
- M-KUBOSはEdge用途にはちょっと高価すぎる
  - 最近は半導体不足、円安でMiddle ClassのFPGAでも高い
- もっと安価なチップをShell-Role型で作ろう

→　SLMLET

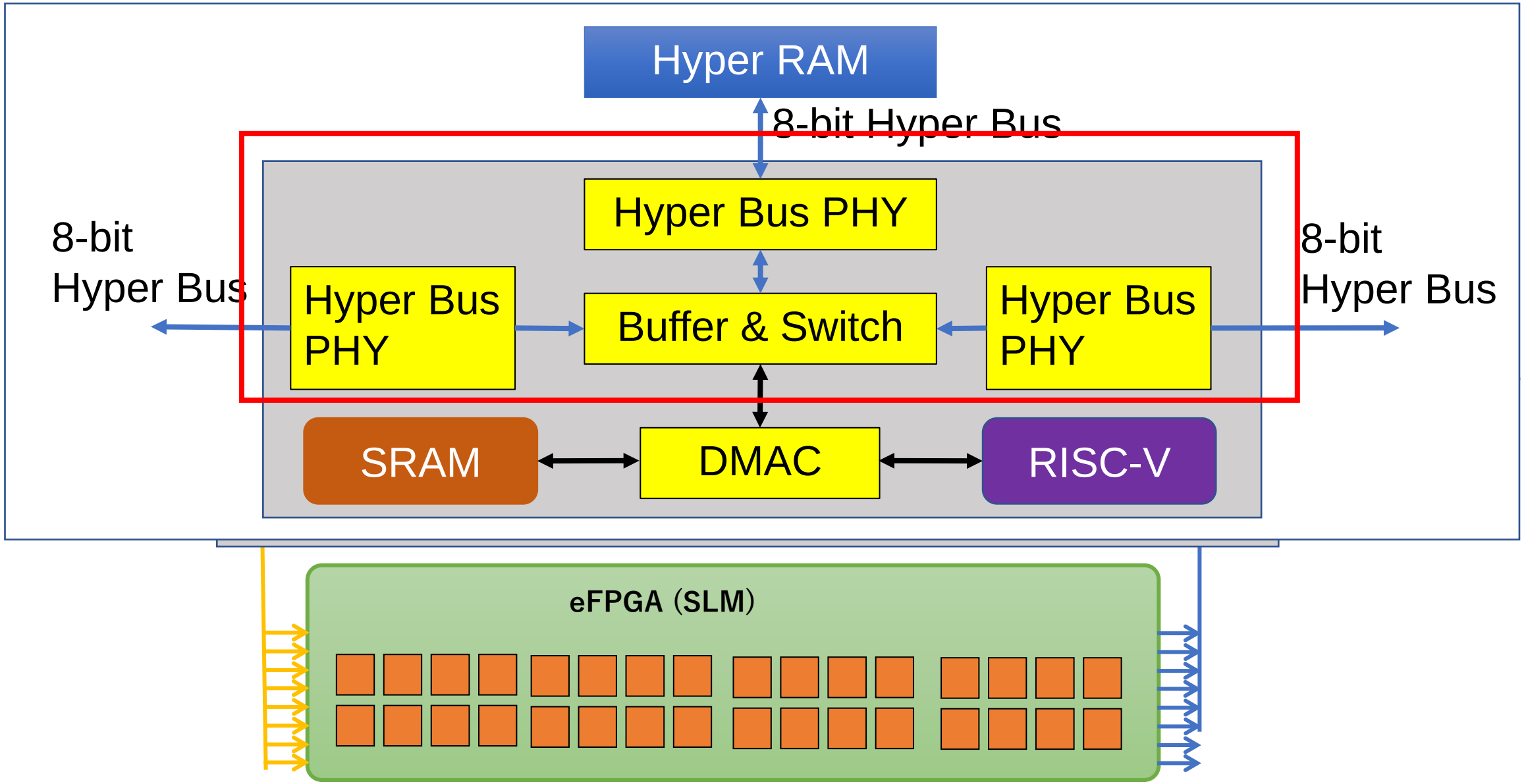
  - 対象アプリケーション: encryption, decryption, anonymization, MQTT off-loading, image filters….
  - Shell partは標準構成：RISC-V、HyperBus、メモリ
  - Role partはオリジナル組み込みFPGA：SLM (Scalable Logic Module)
    - とりあえずSLMLET-1ではSLMは小容量
    - 小容量のFPGAを最大限に利用しよう！
      - HyperBusのネットワークでマルチチップ化
      - Configuration dataの並列高速再構成
      - Configuration dataの圧縮、自動伸長
      - ハードウェアコンテキストマイグレーション
    - Body Biasを用いてFPGAの漏れ電流と性能のトレードオフを実現

Shell part: Processor

We Selected RISC-V

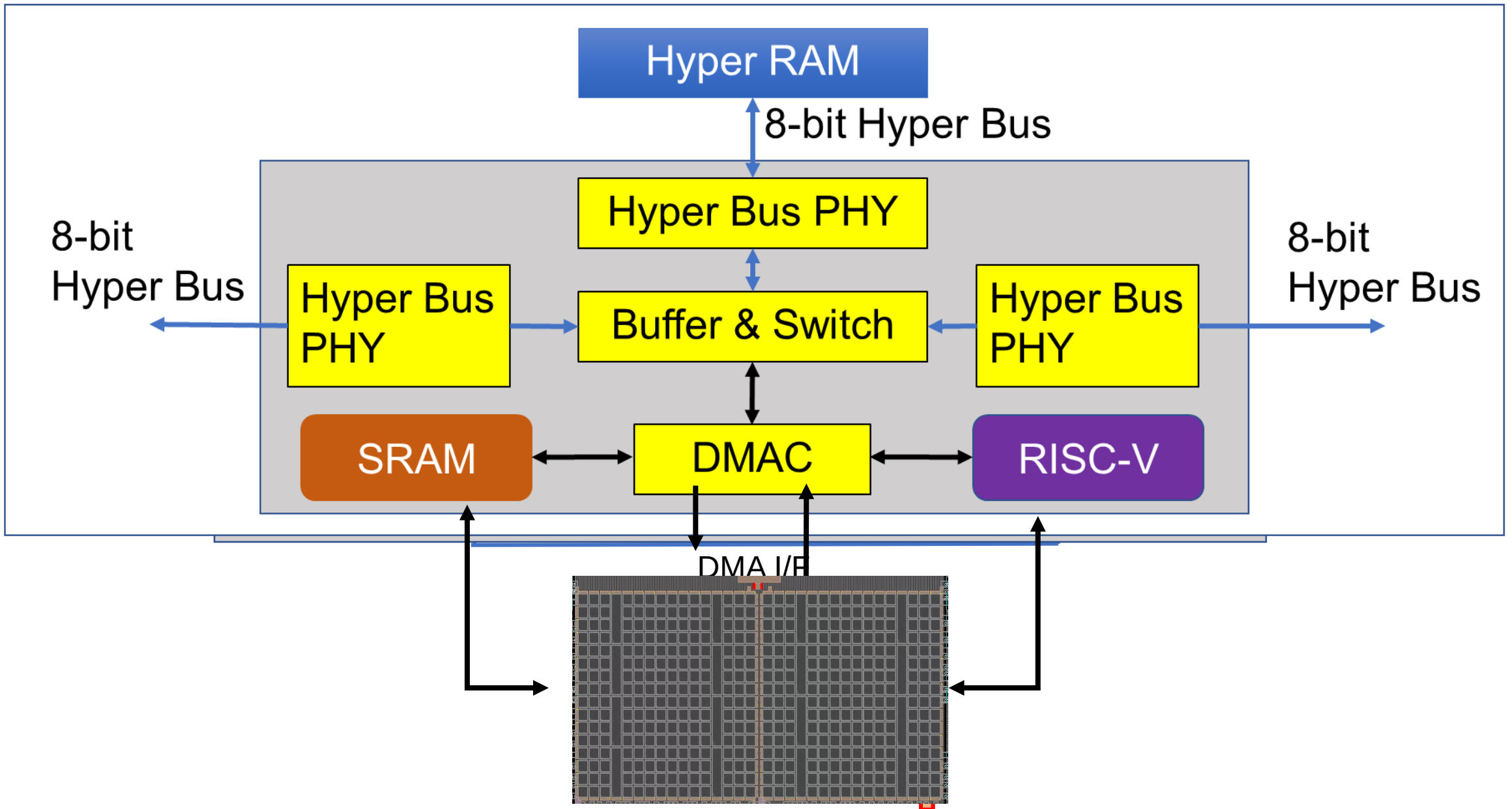


Designing Shell part: Interconnection and Memory



Designing Role part:

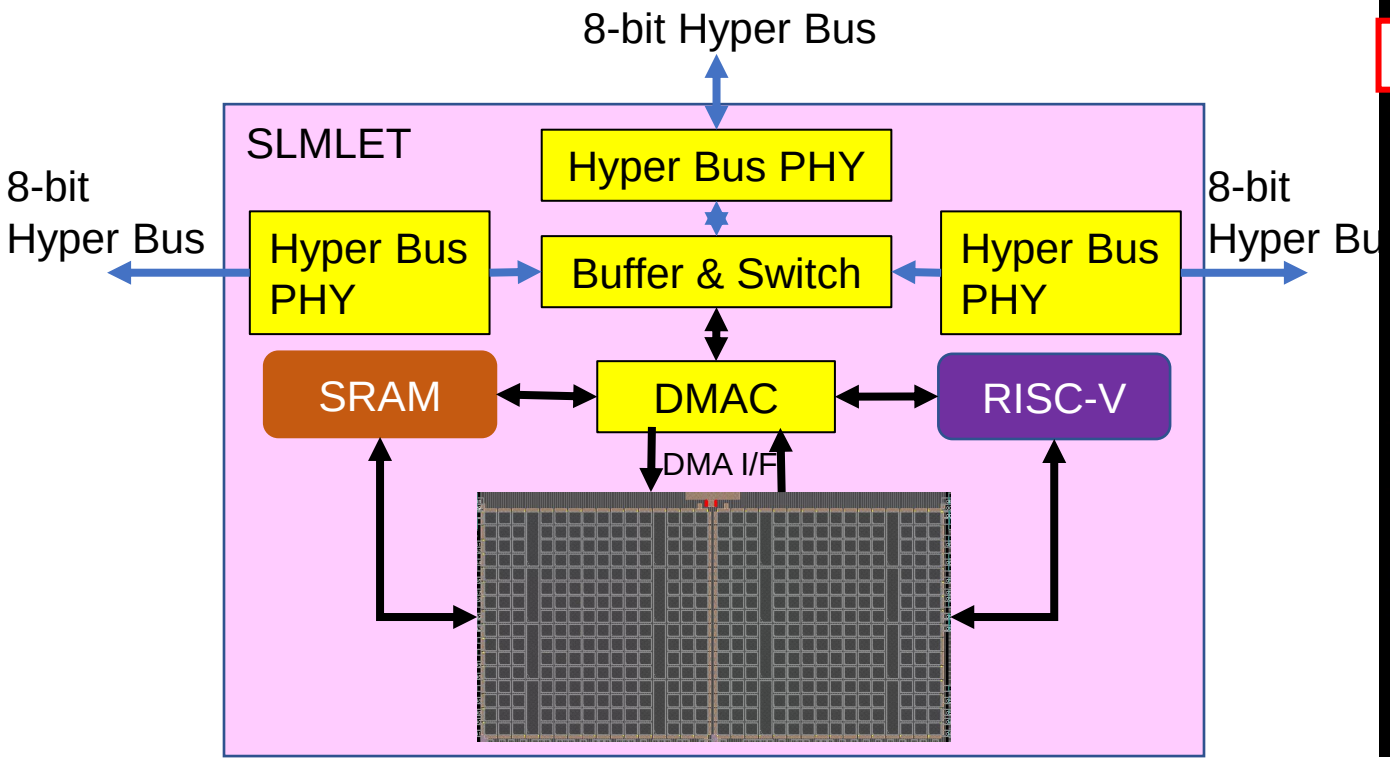
We Selected RISC-V

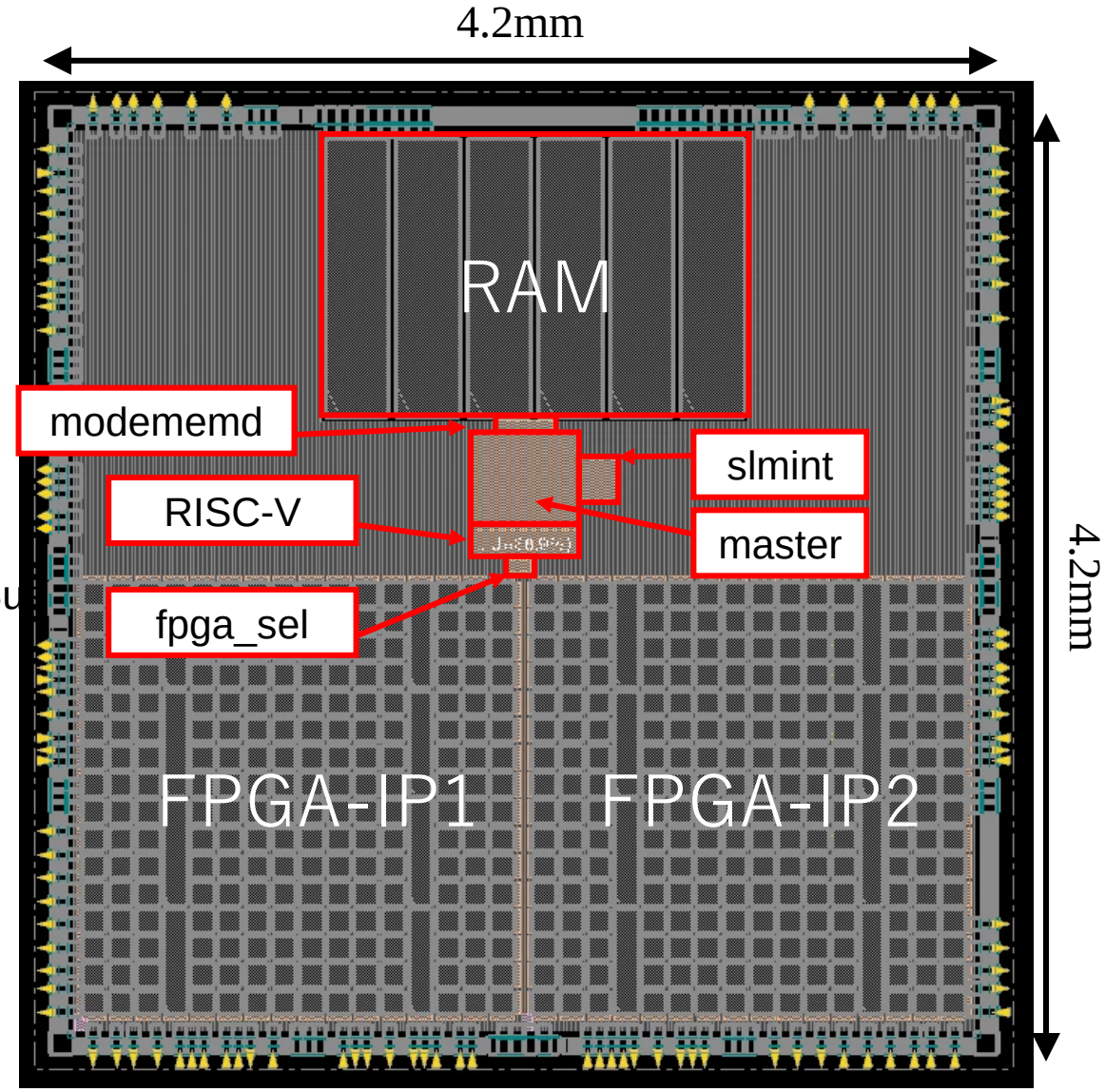


SLMTET-1チップ

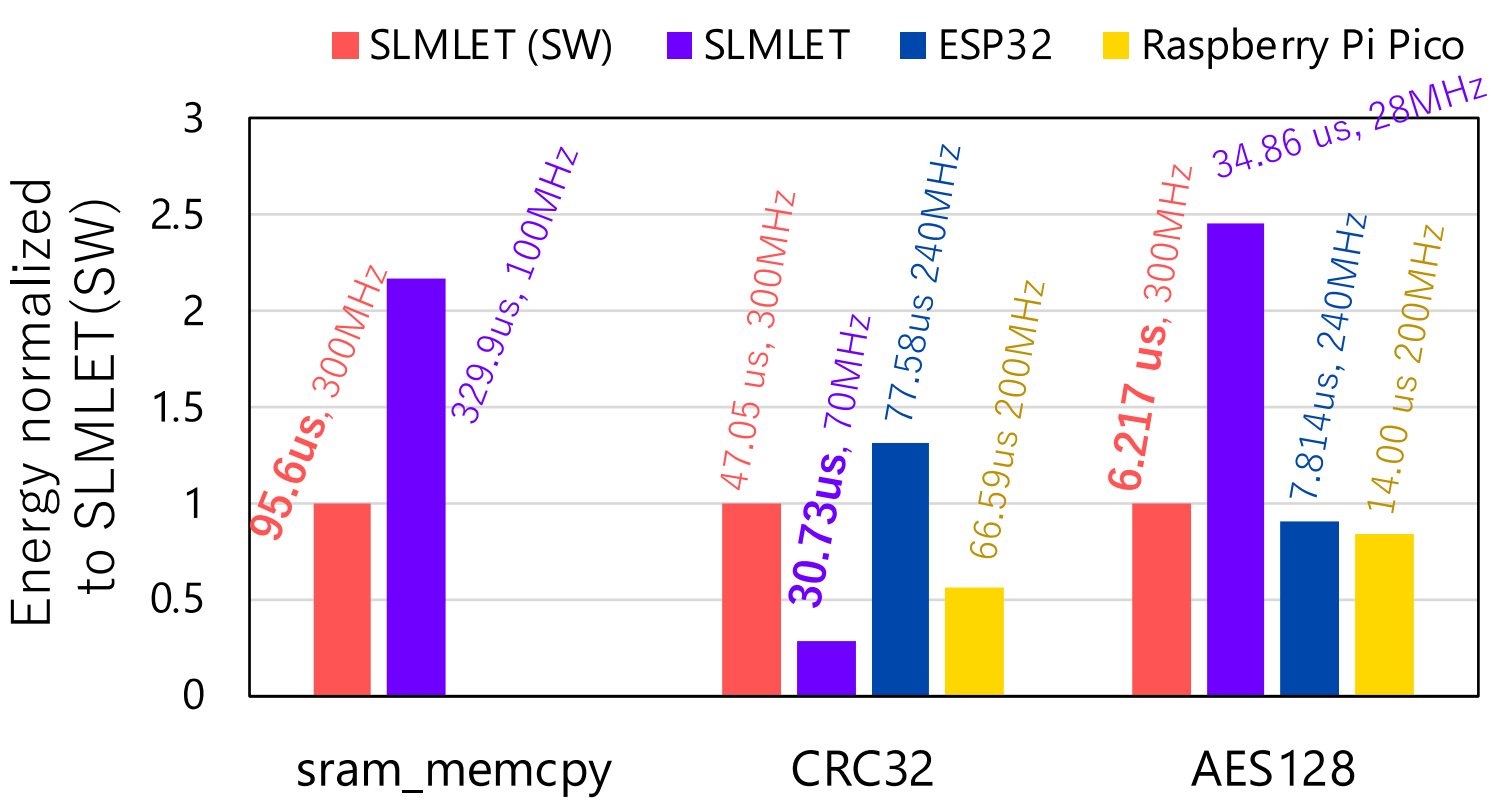
- 小さな面積を有効利用
- Hyper Busでマルチチップ接続が容易
- 内部でコンテキストを複数持ち高速切り替え可能
- ハードウェアコンテキストマイグレーション
- 構成情報の圧縮
- ボディバイアス制御

- ダイサイズ：4.2mm□
- 半導体プロセス：USJC 55nm LVT

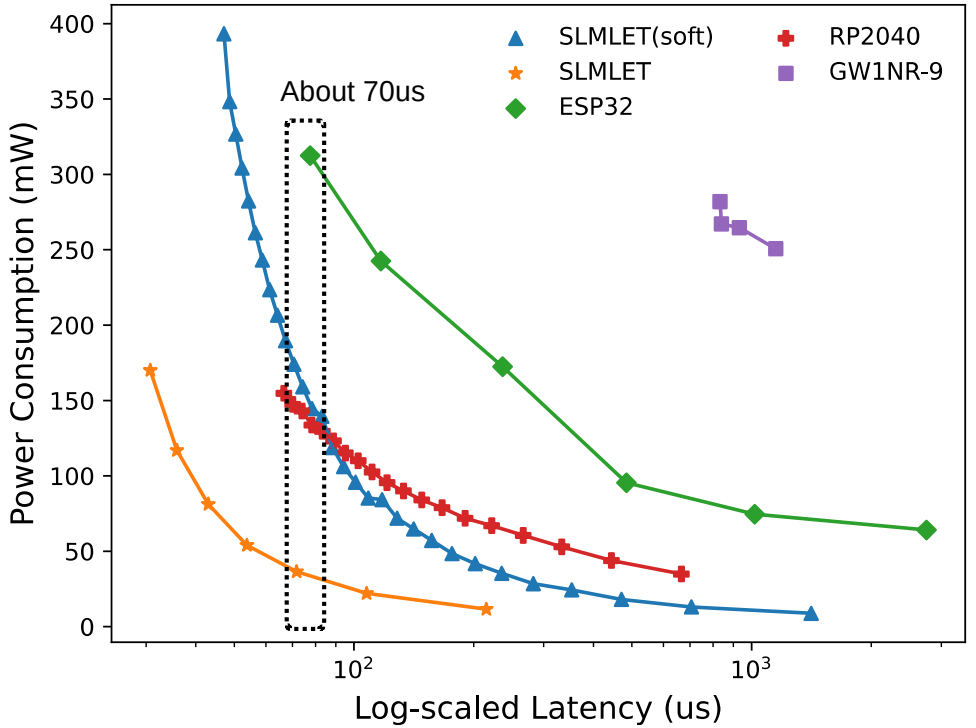




SLMLET-1の評価（CoolChips24）



Energy consumption at peak performance



Power consumption for each frequency

- HW-enabled design archives best energy consumption for CRC32
- However, it limits performance for sram_memcpy because of 16bit width memory I/F and AES128 because of SLM resource limitation and long critical paths.
- Software-only design for AES128 still outperforms the other two commercial micro-controllers (ESP32 and Raspberry Pi Pico)

2番目のシステム

- 「人月の神話」Frederic Brooks Jr.
  - 一般的に、二度目のシステムはデザインし過ぎる傾向がある。最初のシステムデザインでは注意深く外したアイデアと装飾を使ってしまう。

- SLMLET-2は2番目のシステムの罠に陥っていない（陥ることができていない）
  - SLMLET-1の問題点は：
    - いろいろテクニックを実装したが、補いきれないくらい全体が弱体だった
      - SLMが弱体、2つに分かれているので設計がし難い
      - RISC-Vもキャッシュがないので搭載可能なプログラムが制限された
    - マルチチップでも単独チップで搭載できるロジックが小さすぎて拡張が難しい
    - Shell部とRole部の接続がアドホックでバグが発生
      - SLMはRISC-V、HyperBusとの間に3種類のインタフェースを持っていたが、どれも仕様が異なり使い難く、バグもあった

→SLMはモノリシックにし、容量をできるだけ大きく
RISC-Vはキャッシュをつけて性能を増強
標準I/O　IOBでインタフェースを統一

SLMLET vs. SLMLET-2

	SLMLET	SLMLET-2
SLM	16×16のFPGA　IP×2	32×24のFPGA IP×1
Configuration Data　(bit)	103,880×2	323,632
内部メモリ	32KB×2、16KCPU専用	16KBx2way Data/Inst cache, 4KBCPU専用 16KB通信用メモリ
RISC-V	miniRISC-V	cv32e40s
Pipeline	2-state	5-stage
Process	USJC_C55DDCT07 ボディバイアス制御付き	USJC_C55LPT07 ボディバイアス制御なし

諦めたもの：

- チップ内部メモリからの並列Configuration Data転送、Configuration Data圧縮
  内部メモリに対してConfiguration Dataが大きく、複数セット入らない
  FPGA IPが1個なので並列転送はできない
  その代わりHyperRAMからの高速転送は可能→実用上は使いやすい
- ボディバイアス制御
  - C55DDCTの製造が終わってしまった！
  - 副作用としてSRAMマクロが小型化した

Forgotten techniques in commercial FPGAs

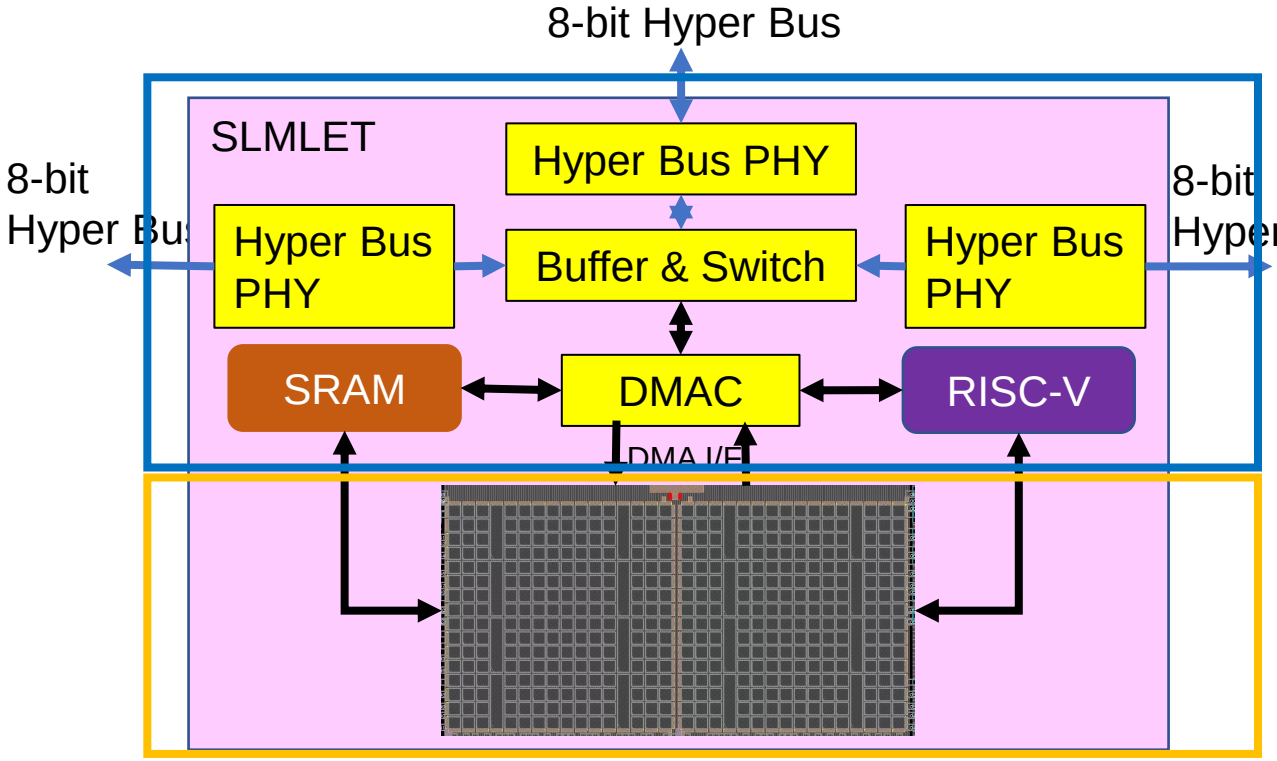
- There are techniques which have been studied long time, but never integrated in commercial FPGAs.
  - Quick configuration exchange with multiple sets of configuration data inside the chip.
  - Configuration data compression and run-time decompression.
  - Hardware context switch /Context migration.

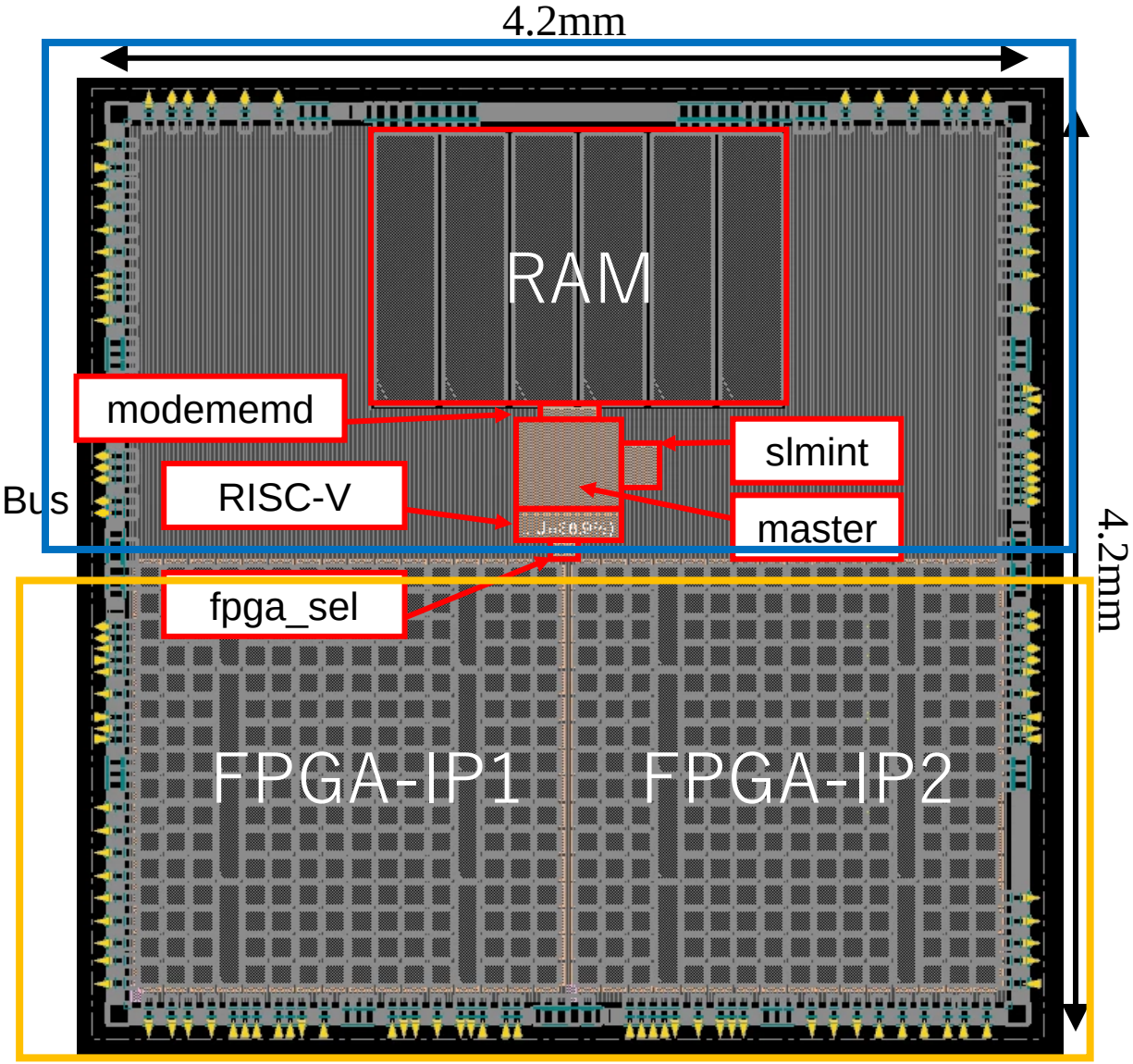
They were implemented in SLMLET and we can try them.

The evaluation results were presented in CoolChips 2024.

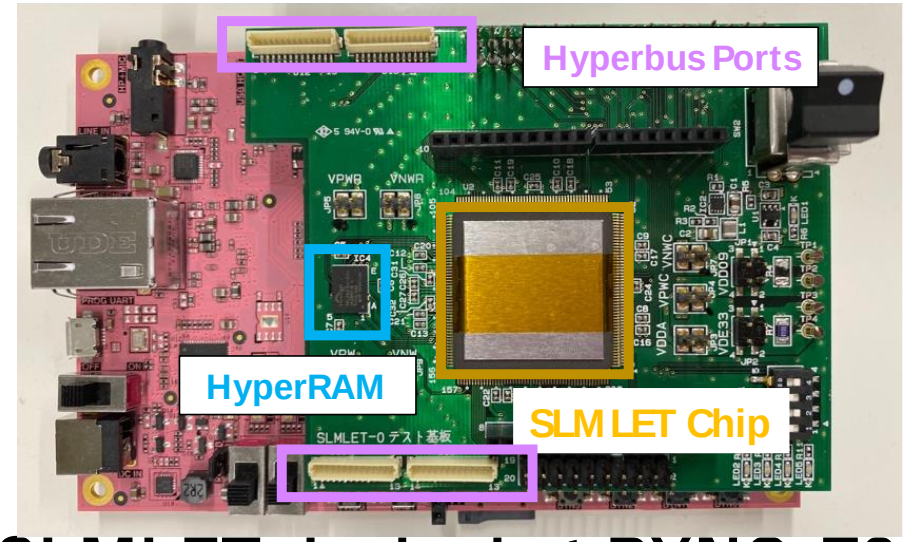
The layout of SLMLET

- Crust part is placed on the upper half, Core part is placed in the lower half.
- DMA, RISC-V, and switch is placed in the center of the chip.
- 4.2mm□ USJC 55nm LVT

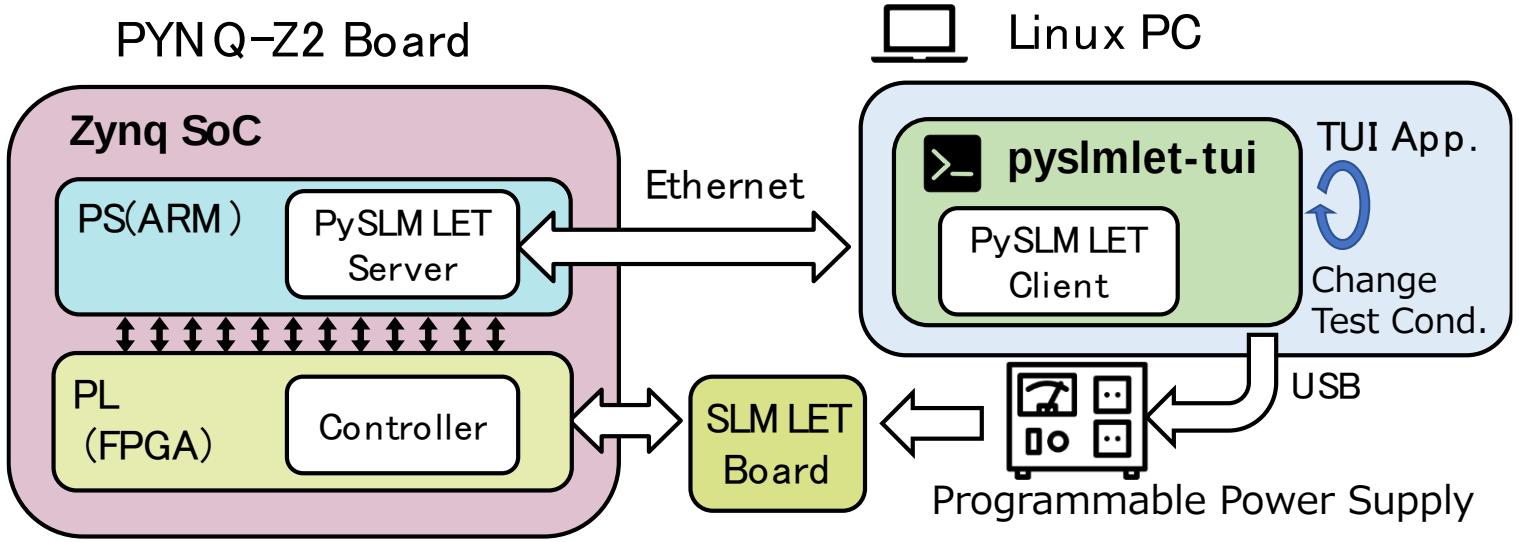




Test & Evaluation Environment



SLMLET docked at PYNQ-Z2



Fully automated test and evaluation

- PySLMLET
  - PYNQ-Z2 board works as SLMLET host
  - Developed Python driver to control HW on PYNQ-Z2 board
- PySLMLET-TUI for test automation
  - Text-based UI for testing with variable conditions (running software binaries, voltage,
    operational frequency, etc.)

Evaluation benchmarks

- Used three applications
  1. *sram_memcpy*
     - Copies 16 KB data block on the shared SRAM
  2. *CRC32*
     - Computes CRC32 value for the given 1KB of binary data (polynomial 0x04C11DB7)
  3. *AES128*
     - Encrypts one block of 128-bit plain texts
- SLM block part is designed with RTL implementation
- Software-only cases uses MiBench source codes

Summary

- We focus on novel MEC systems.
- Now, SLMLET-2 providing more powerful FPGA/CPU is tape-outing.
- Another Shell-Role type chip: Agile-X PF based on minimal fab. is under developed (Another project)

MEC(Multi-access Edge Computing)

5G/6G : low latency (< 0.5msec) and high bandwidth
→New applications treating timing-critical multiple jobs.
Low energy and low cost

Investigating Systems for MEC

From chip to architecture

Smart city application

H.Nishi
Keio U.

Poster/demo

Edge Device | Edge Device | Edge Device | Edge Device

5G Radio

5G Control Plane

Shell-Role Chip | Shell-Role Chip
Shell-Role Chip | Shell-Role Chip
Shell-Role Chip | Shell-Role Chip

Design Tool
High level synthesis

K.Wakabayashi
U.of Tokyo

Distributed OS
Job manager

M.Sugaya
Shibaura Tech.

Poster/demo

Chip/System Architecture

M.Iida
Kumamoto U.

H.Amano
U.of Tokyo

Cloud

5G User Plane

Multi-Access Edge

菅谷G

ROSを活用するインテリジェントロボットシステムの研究 ～介護者を支援するロボット～

菅谷みどり（芝浦工大
大川猛（熊本大学），
佐藤未来子（東海大）

背景

■ 高齢化による労働人口減少に伴い、介護施設においてロボットやシステムによる労働負担軽減が期待される

介護施設「絹の郷」

介護施設でロボットに期待される機能

- **精神ケア**：寝たきり，居室に篭る高齢者へのストレスケア
- **転倒検知**：転倒した入居者の早期発見
- **物の運搬**：介護士が入居者を見ている間に物を運ぶ
- **顔検出と挨拶**：食事の時間に入居者を迎えに行く

課題

■ 人の気持ちを理解する手法 ➔ **生体情報による感情推定技術**

■ 介護ロボットに求められる画像処理には高い処理性能が必要

■ FPGA(Field Programmable Gate Array)クラスタ(M-KUBOS) を活用した並列処理

■ 状況に応じたロボットの複雑な振る舞いを実現することが必要

■ Robot Operating System(ROS)で実現する分散並列処理

「絹の郷」スタッフ○○さん

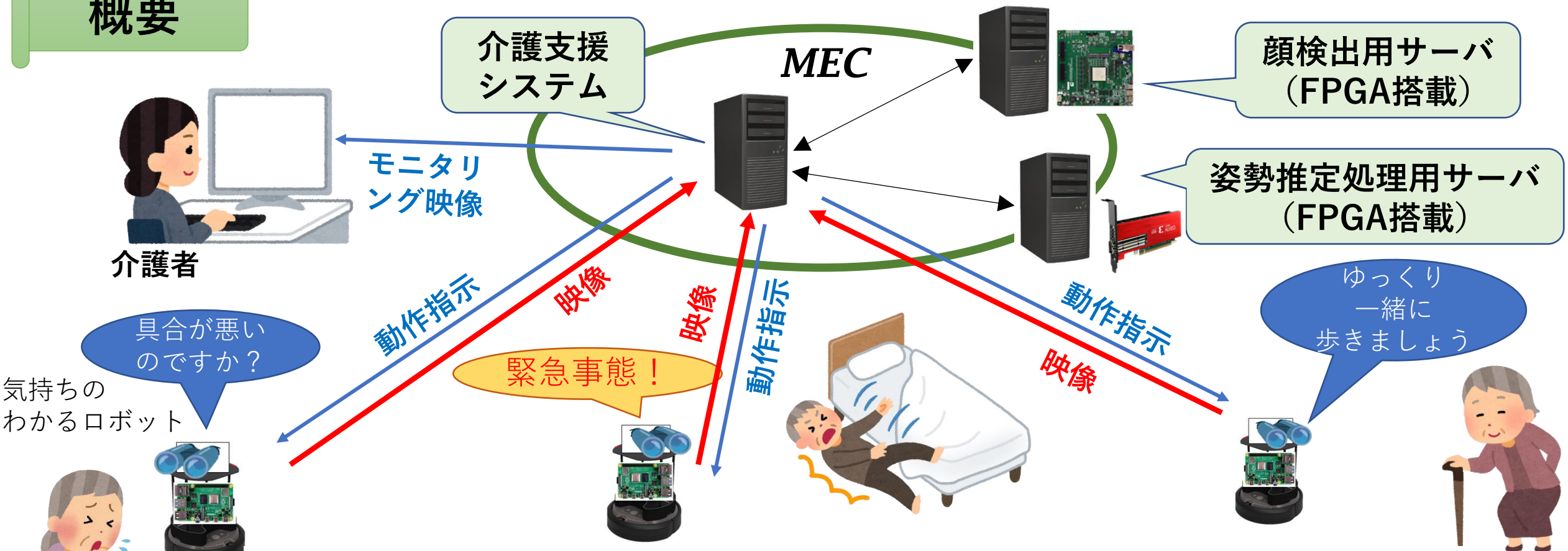
ROSを活用するインテリジェントロボットシステムの研究
～ロボットで介護者を支援する～


東海大
佐藤未来子
准教授


熊本大
大川猛
准教授

システム概要

■ Multi-access Edge Computing(MEC)：サーバ管理とジョブ依頼処理

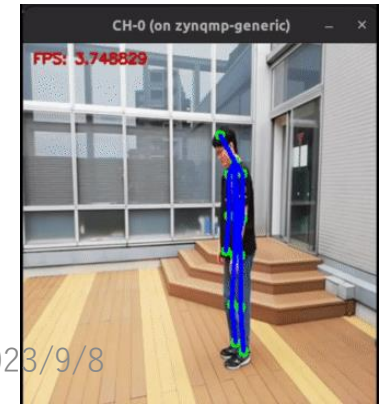
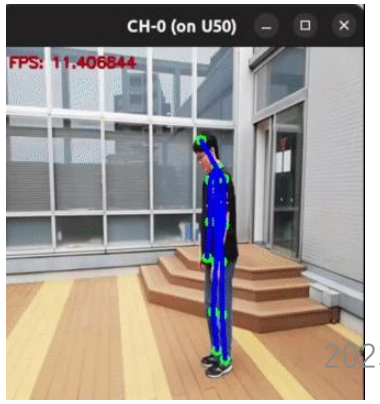


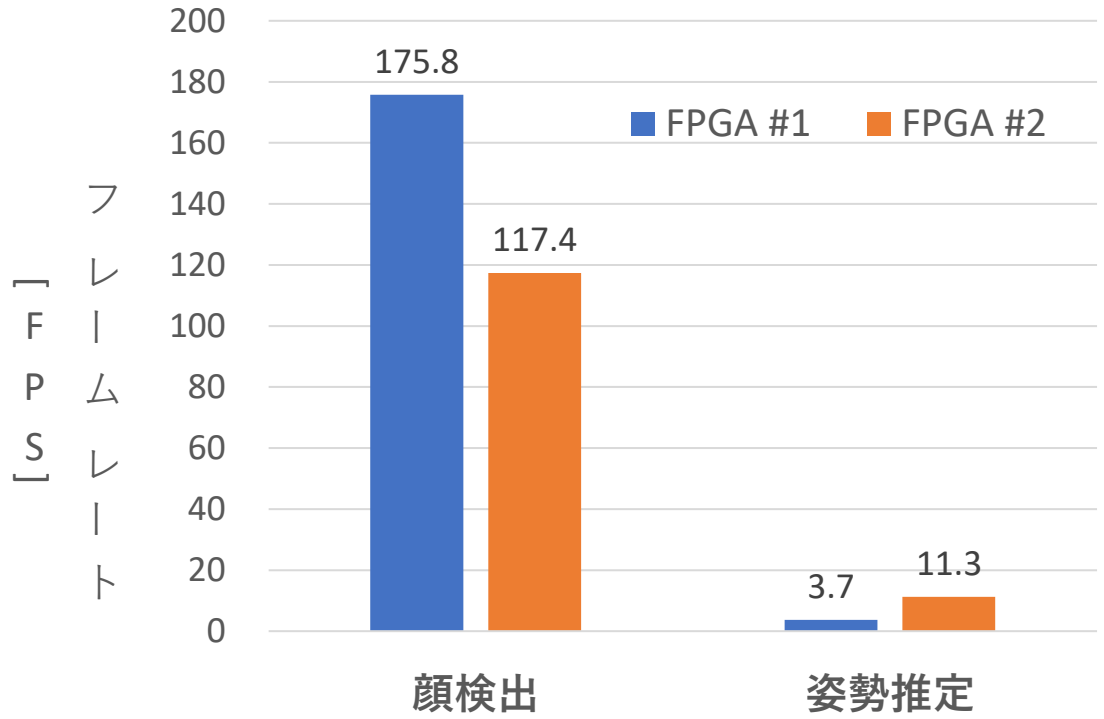
■ インテリジェントロボット：入居者の撮影、動作指示に応じた自律制御（声掛けや居室移動）、ROS 2で実現

ROSを活用するインテリジェントロボットシステムの研究 ～ロボットで介護者を支援する～

FPGA性能

- FPGAの種類により姿勢推定と顔検出の処理性能に差がある
- MEC内で適切なサーバリソースを活用する

	FPGA #1	FPGA #2
顔検出		
姿勢推定		



【FPGA #1】M-KUBOS
【FPGA #2】Alveo U50

ローカル5G利用時のネットワーク構成図

*実は5Gレピータ（ネットワーク中継機器）を経由している

地下実験室
3 か所に配置
されている

ローカル5G設備詳細

GPSアンテナ
屋上

無線装置制御用
サーバー(DU)

同軸ケーブル

この回線は
認証用

ONU

クラウド5GC接続用
ネットワーク機器

L3SW

クラウド5GC
/ SIM認証

フレッツ光ネクスト
監視専用
インターネット経由

無線装置制御用
サーバー(CU/UPF)

5G通信*

ローカル5G基地局装置
(IndoorRU: Radio Unit)

5G対応スマホ
で動画を送付
する手もある

5G通信

5G

5G対応
スマホでテ
ザリング

ローカル5G端末
(UE:User Equipment)

5G

GWルータ

学内FW経由で
インターネット
接続回線

姿勢推定処理用サーバ
（AlveoU50搭載）

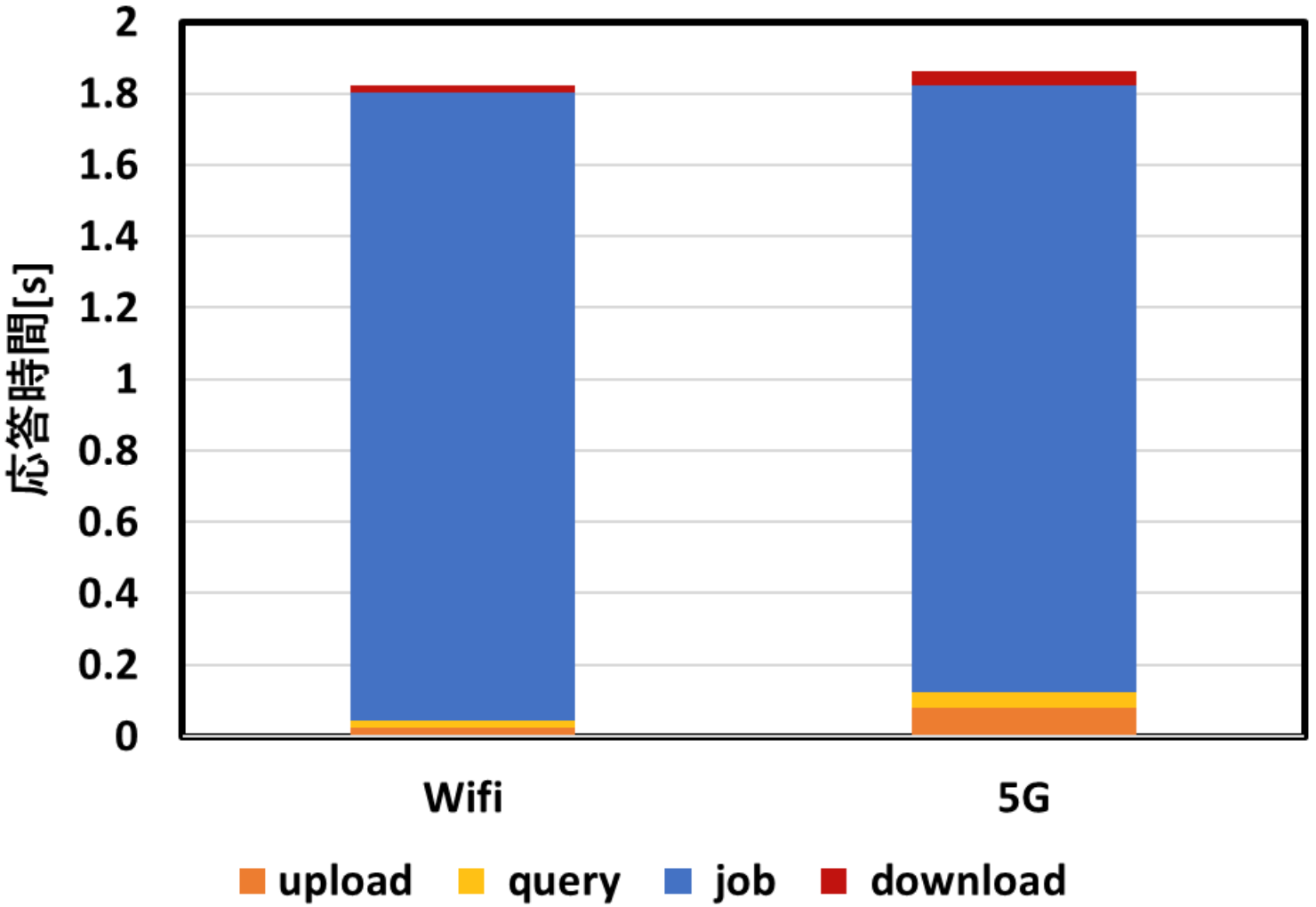
顔検出用サーバ
（M-KUBOS）

MEC-server

（MEC-worker）

Raspbery Pi 4B
（MEC-requester、結果表示機能）

結果

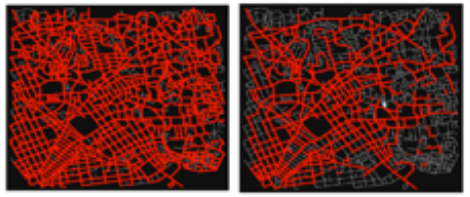


WIFIと応答時間は変わらない➔MECとB5Gの連携部分分析と改善

西G

M-KUBOSによるエッジストリーム匿名化

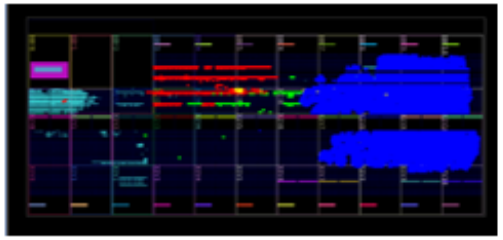
- スマートシティ情報インフラでは、レンタサイクルやスマートフォンなど大量の位置情報を収集・提供する際、必要な匿名化レベルで、すぐにサービス利用したいというニーズに対応
  - M-KUBOSによる経路キャッシュ型匿名化HW化により従来の2桁高速化
  - 一方で情報の劣化率(Information Loss指標:IL)をほぼ変化させずに達成
  - 目標スループットは、2kmメッシュ位置情報の1時間情報量を基準に>100,000 件/s



k=4　　k=20

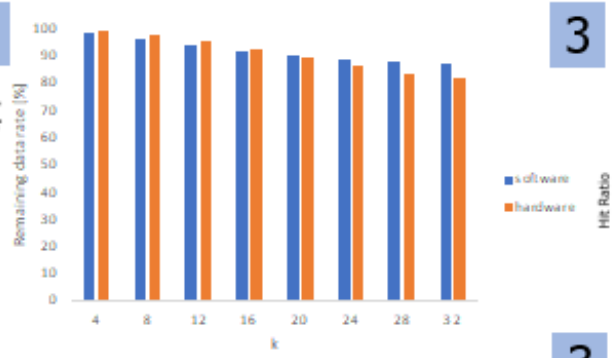
- 結果
  - 1 1km²(密)におけるデータ残留率はソフトウェア同等
  - 2 スループットはデータ数:45,647件(4km²)処理で
    - M-KUBOS **0.604 s**(+データ転送時間**0.308 s**)

    10万件まではリニア、FIFO深さに達し転送ボトルネック
    - CPU Intel Core i9-10940X 3.30GHz **127s**
  - 3 キャッシュヒット率は約70%で30%の低遅延化
  - 4 回路はメモリネック

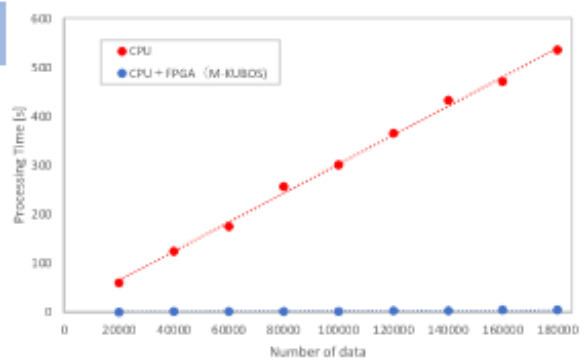


緑:ノード近似
青:経路探索
黄:セグメント生成
赤:匿名化

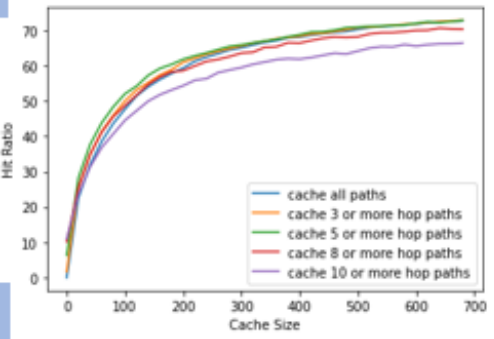
1



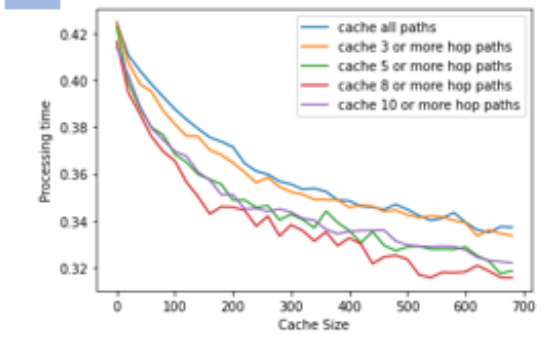
2



3



3



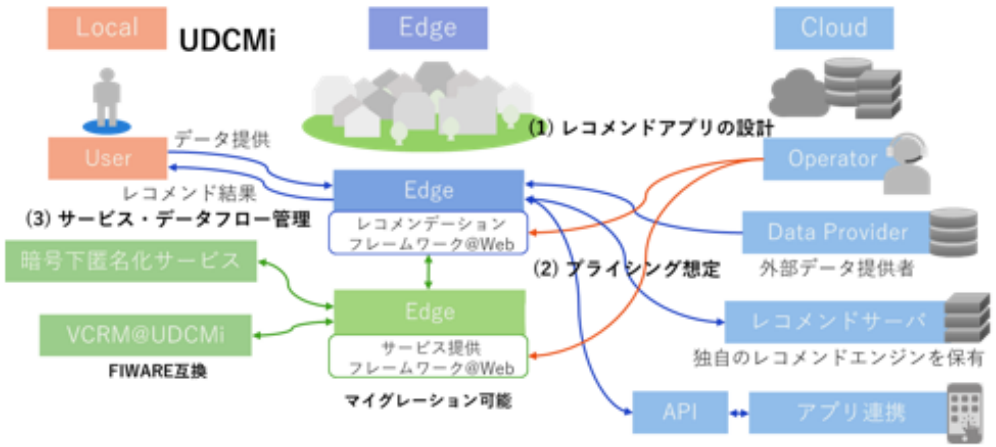
West Lab

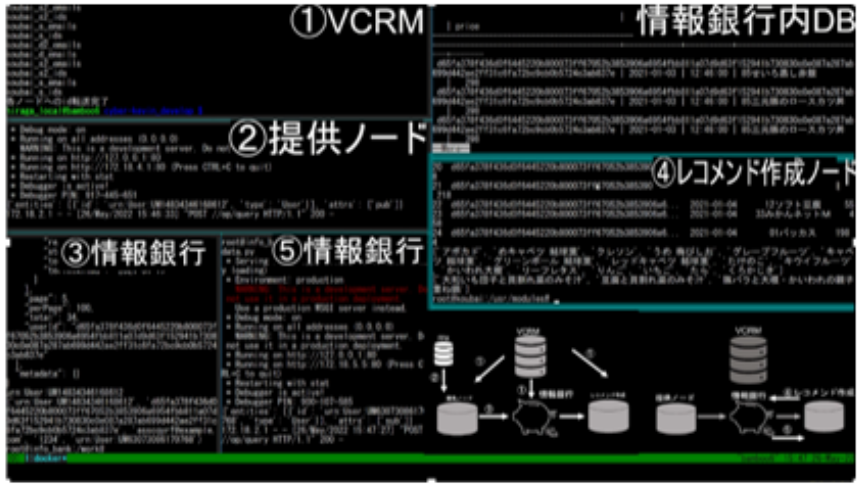
さらなる拡張と応用

- よりILを削減するために
  - k-匿名化は、同じデータ列がk個以上あれば匿名化しなくてもよいという考え方
  - 地図など経路の正確性があいまいであっても、確率(kを小数化)で判断できないか？
- 過去の履歴を用いたIL削減手法
  - 途中経路不明な位置情報列は、忘却曲線重みを付与した位置履歴から確率でkを算出
  - 結果として、人通りが少ないが利用のある通り、大通りではなく、交通を避けて人が通るような道路を正しく表現できるようになったが、現状SW実装のため、計算コストが問題
- スマートシティインフラとしてのエッジHW
  - 情報流通を管理し、正しい利用を見守る際のボトルネックをエッジHWにより解消

提案手法

比較手法





West Lab Computing and Networking

MEC指向HARKベース音響処理フレームワークの構築

HARKの主要アルゴリズム２つのM-KUBOS化

- 音源定位 (Multiple Signal Classification)
- 音源分離(GHDSS)

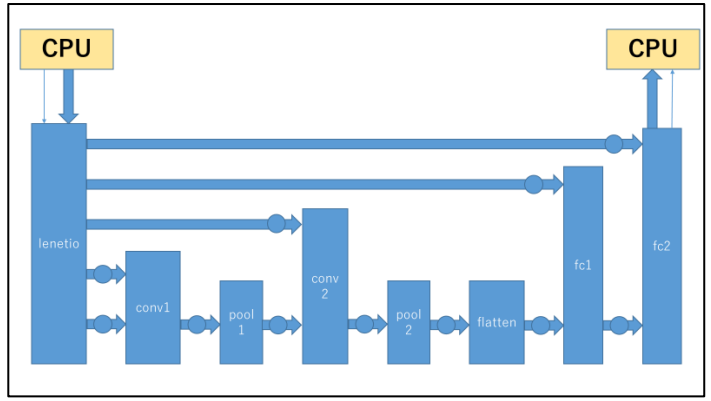
使用リソース、サイクル数、実行時間

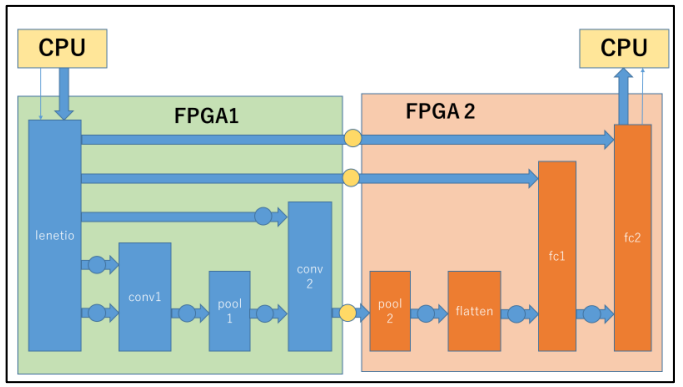
	LocalizeMUSIC		GHDSS	
	最適化前	最適化後	最適化前	最適化後
Cycle	97,396,431	46,964,440	551,958	435,187
実行時間	1.696ms	1.327ms	3.526ms	3.187ms
DSP	189	747	355	502
BRAM	104	326	141	144
LUT	72,029	364,844	83,426	139,450
FF	92,911	343,923	139,450	179,033

最適化により，フレーム処理レベルでの高速化を確認
CPU側処理，メモリアクセス処理を含めたEnd-to-end 処理での最適化を実施予定

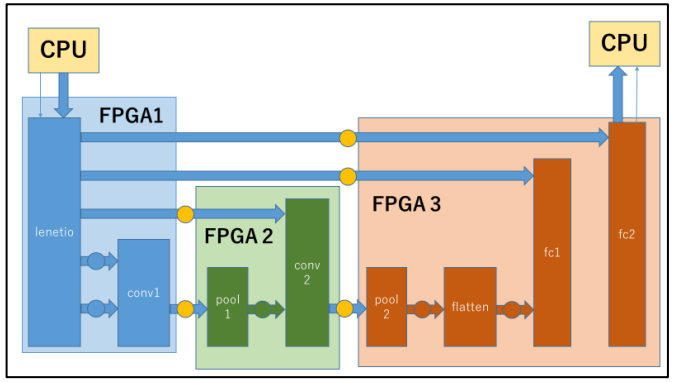
若林G

マルチFPGA設計環境





分割例 1



分割例 2

- 2022年後半の成果

SystemC上の変数、配列を分割点と指定すると、その部分で分割される

分割点には、自動的にストリーム転送回路が挿入される。

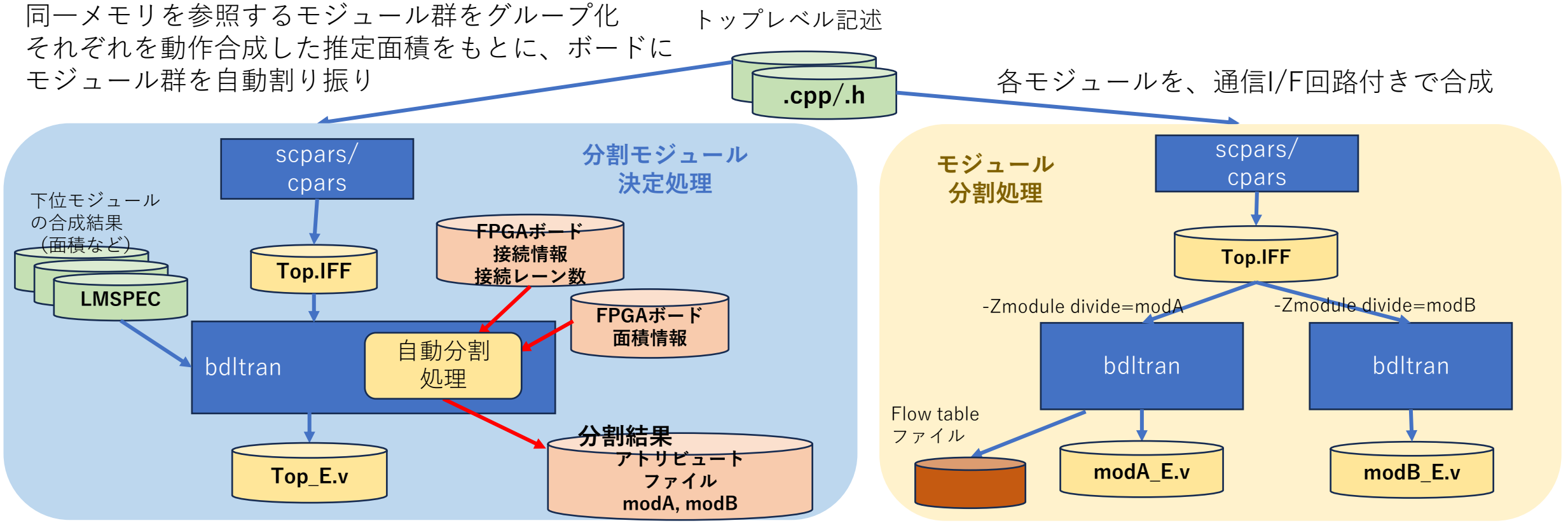
転送回路は、データのパック、アンパック、複数の配列を、一つの配線で時分割多重する、その場合の流量調整、

AIXでのCPUとの通信等を行う回路が自動挿入される

マルチFPGA向け自動分割機能

巨大なSystemC/C++記述を、自動的に複数FPGAボードに分割する機能を作成中
昨年度から基本仕様を検討し、実装検討に入っている。
・FPGAボードに搭載可能なモジュールに分割
・分割点はストリーム化可能で、FPGAボード間で効率的に通信可能な部分
・M-KUBOSの物理的なネットワークを考慮して、できるだけよい分割、通信経路を自動決定

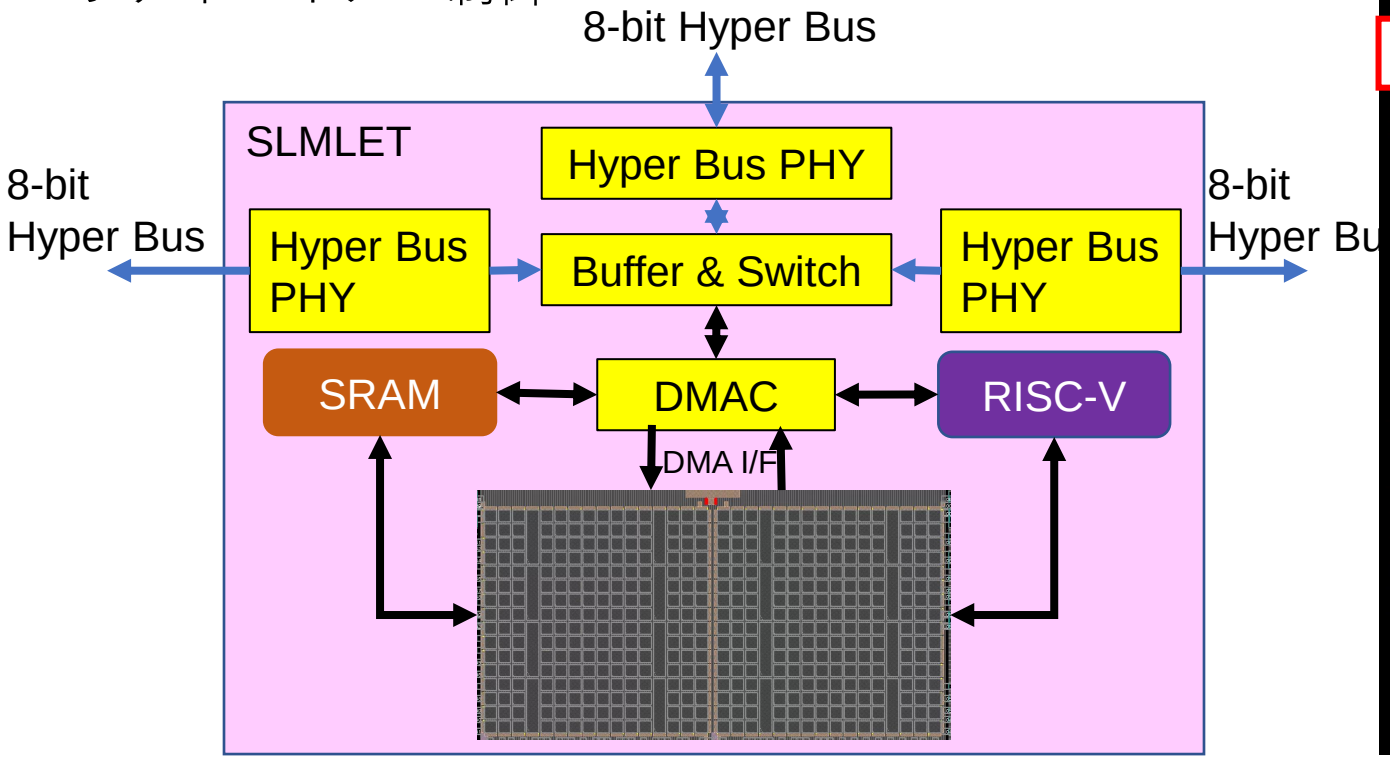
同一メモリを参照するモジュール群をグループ化
それぞれを動作合成した推定面積をもとに、ボードに
モジュール群を自動割り振り

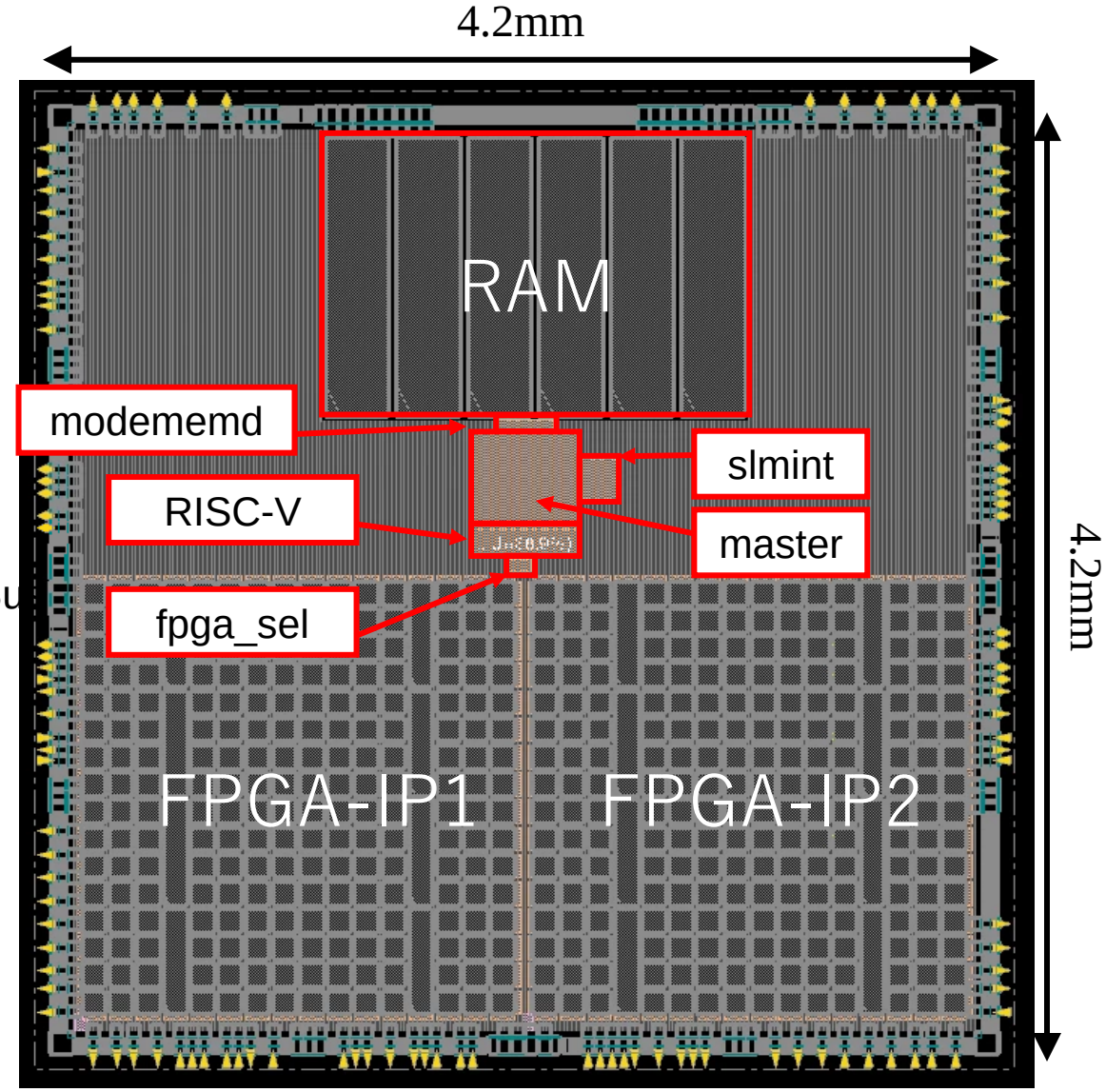


飯田G　次世代低電力FPGA　SLMTETチップ

- 小さな面積を有効利用
- Hyper Busでマルチチップ接続が容易
- 内部でコンテキストを複数持ち高速切り替え可能
- ハードウェアコンテキストマイグレーション
- 構成情報の圧縮
- ボディバイアス制御

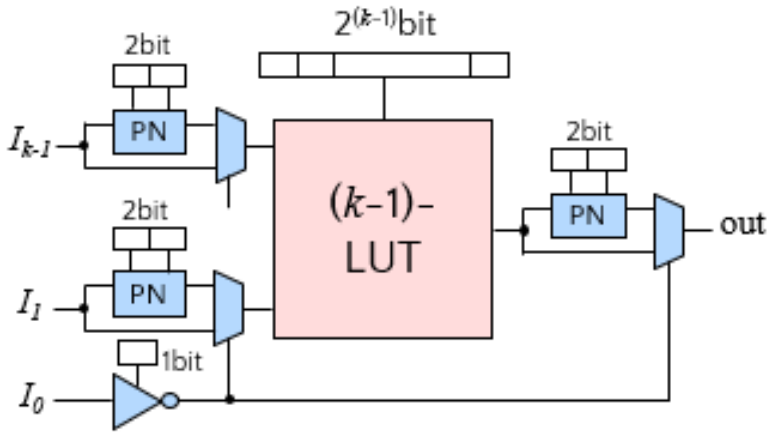
- ダイサイズ：4.2mm□
- 半導体プロセス：USJC 55nm LVT





SLMの構成

- Scalable Logic Module(SLM)
  - 省構成メモリなロジックセル
  - K入力ロジックを(K-1)-LUTとProgrammable NANDによって実現
  - シャノン展開とNPN同値変換を利用
- SLMを用いたFPGAのIP生成
  - SLMの入力サイズ、ロジックセル数などがカスタマイズ可能なフレームワーク[1]
  - 相互接続のスイッチブロックはWilton型



K入力ロジックのためのSLMブロック

SLMLETに使用されたIPのパラメータ

SLM構成	セル数 / クラスタ	クラスタ数	総SLM数	DSPブロック	総FF数
5-SLM	4 BLE (SLM)	224	896	8	1024 bit

[1] Kuga, Morihiro, et al. "An eFPGA Generation Suite with Customizable Architecture and IDE." *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 106.3 (2023): 560-574.

SLMLET向けFPGA-IPの概要

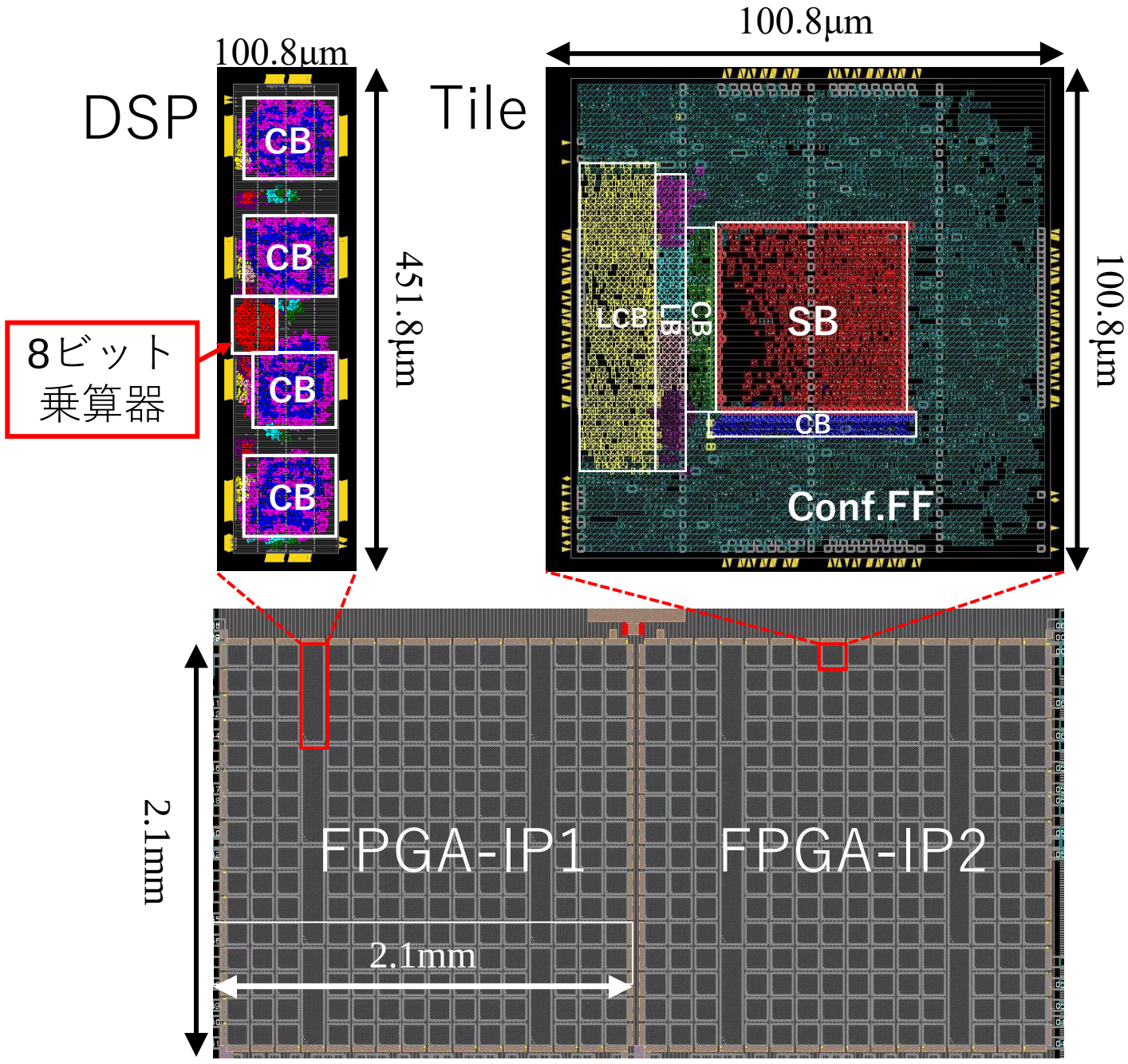
FPGA-IP 1個あたり

- アレイサイズ：16×16
- 論理クラスタ数：　4
- 基本論理要素数：896
- DSP(8bit乗算器)：　8
- I/O端子数：　640

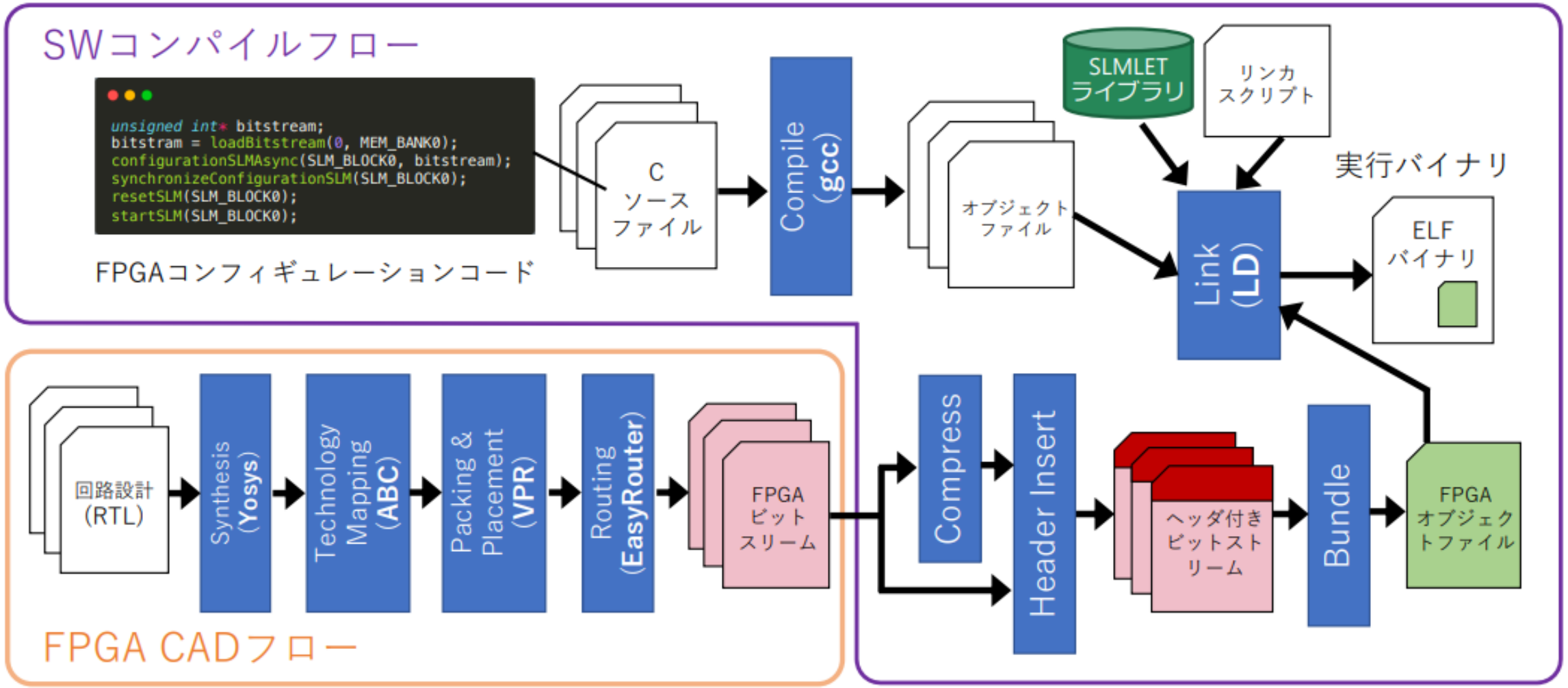
専用演算器の導入により，演算遅延が削減できることをSTA解析により確認

FPGA-IPの相互接続

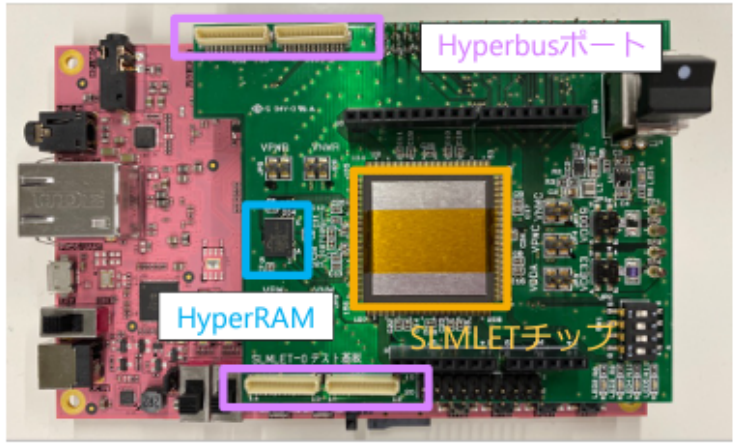
- IP1のE側，IP2のW側を相互接続（IN-OUTペア160組，計320本）
- RISC-V，DMAC，RAM I/Fは両方に接続し，どちらで使うかを制御信号で選択



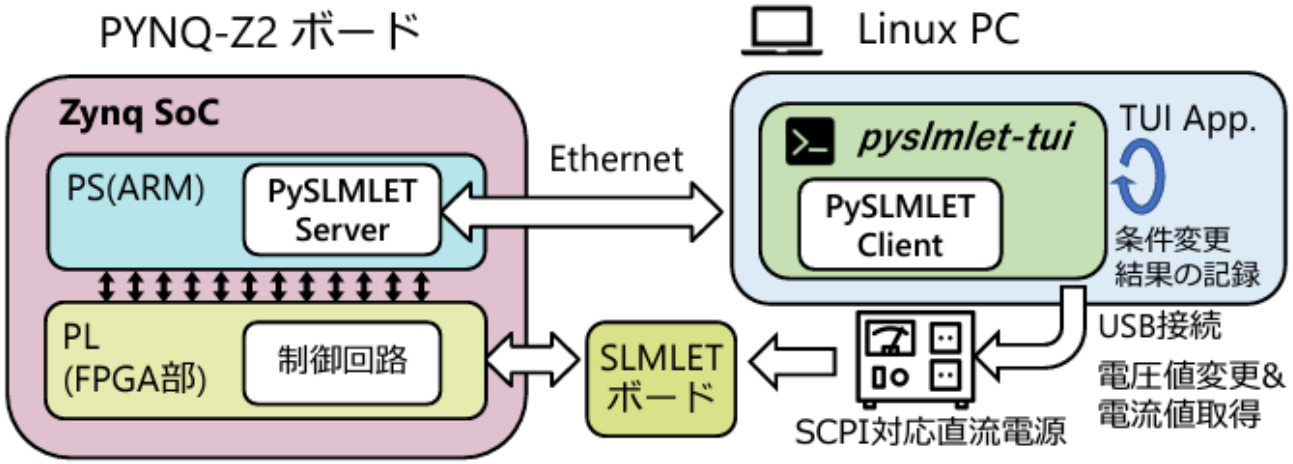
SLMLETチップの設計環境



評価環境：PySLMLET-TUI



SLMLETボードとPYNQ-Z2



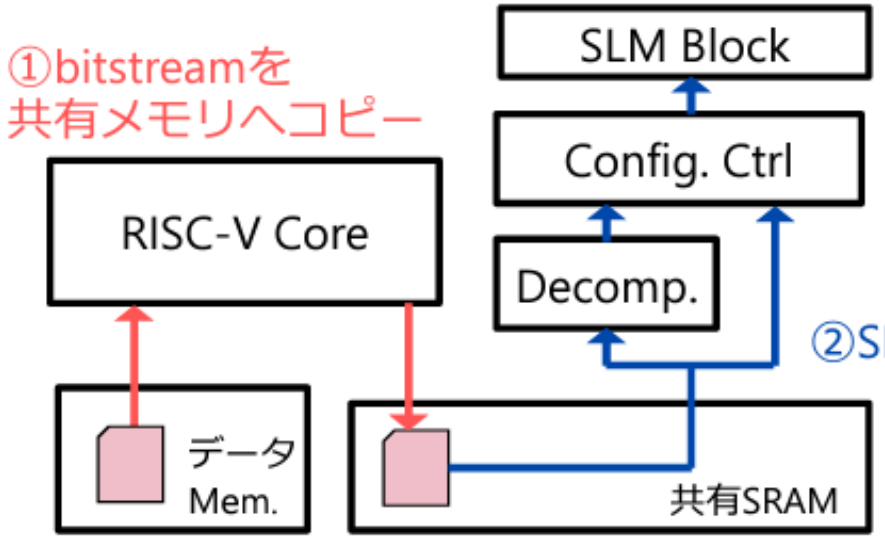
- PySLMLET-TUI
  - 実行バイナリや電圧、周波数など条件を変更しテストを行うText-based UI
  - 簡単なシェルスクリプトで自動化測定を可能に
  - 先行研究でGUI版は実装済み [デモ動画]
- PySLMLET[3]
  - PYNQ FPGAはSLMLETのホストとして振る舞う回路が実装
  - Pythonのドライバでこれらを制御

評価環境の概要

- pynqモジュールのimportに時間がかかりすぎる (20sec程度)
- サーバとクライアントに分離
- TUIはTCP/IPでサーバに接続
- TUI appの起動時間を短縮
- TUIを別マシンでも実行可能に

[3] 小島拓也, et al. "Jupyter Notebook を介した RISC-V SoC 向け実機テスト環境の構築." 研究報告組込みシステム (EMB) 2023.24 (2023): 1-7.

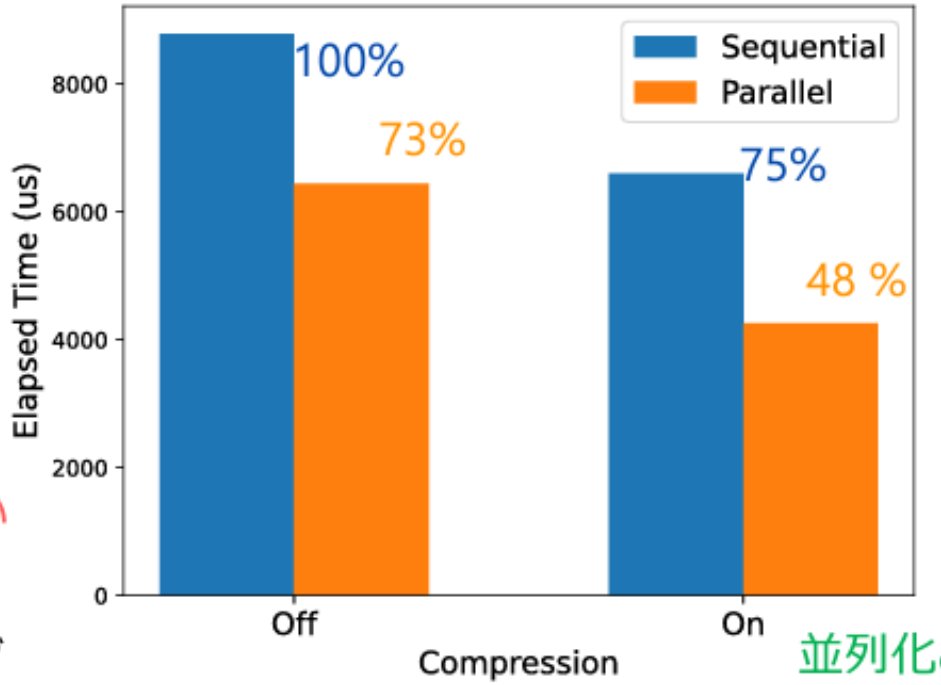
再構成にかかる時間



圧縮してもコントローラがSLMブロックに書き込む時間は同じ

②SLMブロックへ転送

並列化してもSRAMへのコピーは直列
→ 半分にはならない

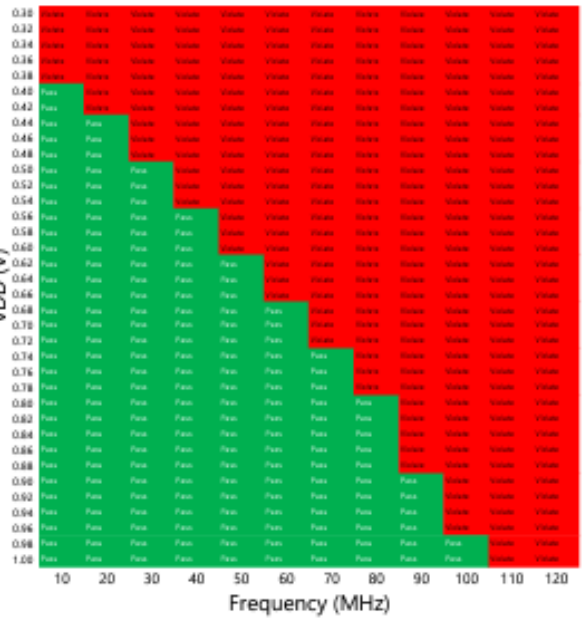


並列化&圧縮によりおよそ半分の時間に

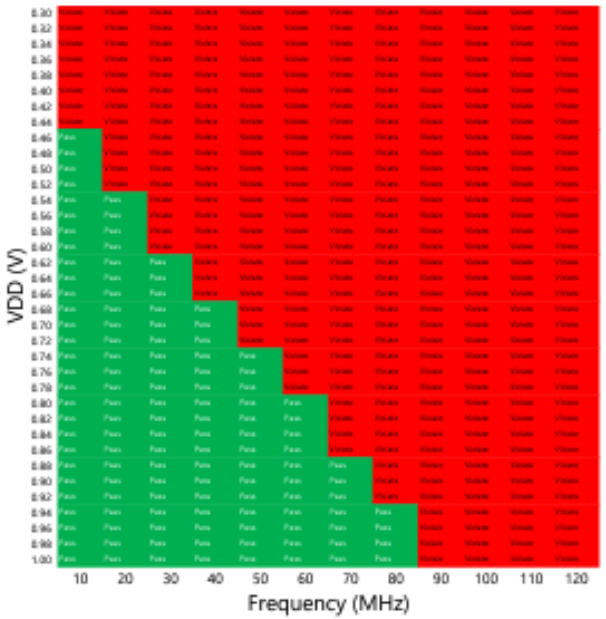
- 評価パターン
  - 二つのブロックを直列 or 並列に再構成
  - 再構成のライブラリ関数 ブロッキング or ノンブロッキング
  - TLCによる圧縮 or 非圧縮
- FPGA上の回路: RISC-Vから読み書き可能なメモリマップトなレジスタ 32bitx4本
  - SLM使用率: 29%, データ圧縮率49%
- 最新の類似研究[4]では6720 6-LUTの再構成に1msだがプロセッサは最大1.1GHz程度で動作
  → SLMLETの再構成時間は十分に高速といえる

[4] Chang, Ting-Jung, et al. "CIFER: A 12nm, 16mm 2, 22-Core SoC with a 1541 LUT6/mm 2 1.92 MOPS/LUT, Fully Synthesizable, CacheCoherent, Embedded FPGA." *2023 IEEE Custom*

SLMブロックの動作範囲

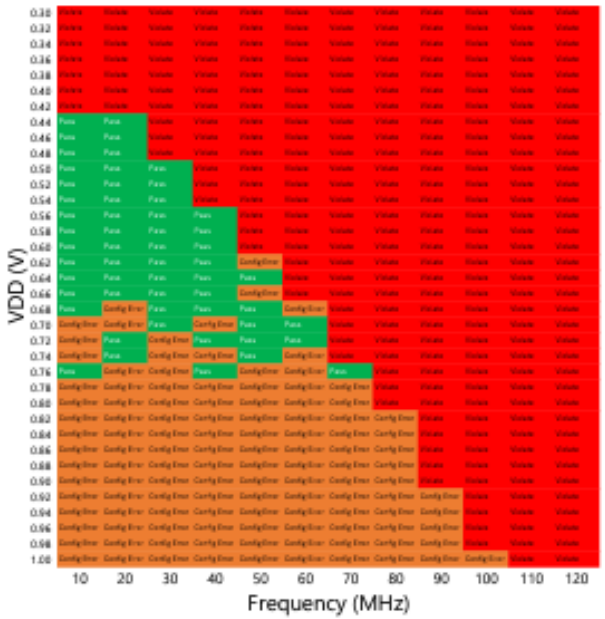


VPW=0 V (f_{max} = 100MHz)

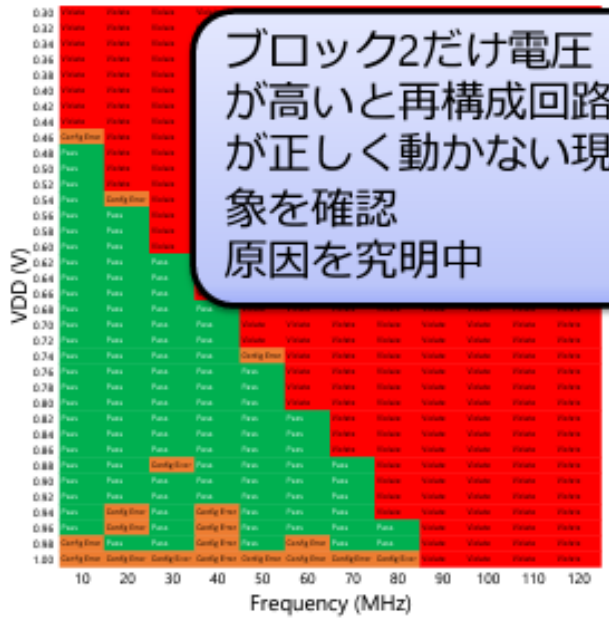


VPW=-0.4 V (f_{max} = 80MHz)

ブロック1



VPW=0 V (f_{max} = 100MHz)



VPW=-0.4 V (f_{max} = 80MHz)

ブロック2

ブロック2だけ電圧が高いと再構成回路が正しく動かない現象を確認
原因を究明中

- 2つのブロックをそれぞれ正しく再構成および計算ができる電圧、周波数を測定
  - 先ほど同じメモリマップトレジスタの回路を使用 (最大SLM段数 6)
  - 電圧: 0.30-1.00 V (標準 0.90 V), 0.02 V刻み
  - 周波数: 10-120MHz, 10MHz刻み
  - ボディバイアス電圧: 0.0 V, -0.1 V, -0.2 V, -0.3 V, -0.4 V
- 動作特性は概ね同じ、想定よりも高い周波数でも動作した

失敗

失敗

再構成だけ失敗

他のシステムとの比較

	SLMLET (本研究)	ESP32	Raspberry Pi Pico	Tang Nano 9K
実装プロセス	USJC 55nm	TSMC 40nm	40nm	TSMC 55nm
CPU	RV32I	Xtensa LX6	ARM Cortex-M0+	N/A*
定格クロック	N/A	240MHz	133MHz	N/A
FPGA	5-SLM 896 x2	N/A	N/A	4-LUT 8640
使用した開発環境	SLMLET SDK	ESP IDF 5.0.2	Pico SDK 1.5.1	Gowin 1.9.8.11 IDE
GCC	12.2.0	11.2.0	10.3.1	N/A
備考	チップ電力を測定	ボード電力を測定 通信モジュールはプログラムから明示的にOFF	ボード電力を測定 200MHzまでオーバークロックで動作を確認	ボード電力を測定 FTDI D2XX Driver 1.4.27

*ボード上に別チップとしてBL702(RISC-V,144MHz)が搭載しているが基本はUART,JTAG用

- SLMLETはRISC-Vコアのみ&SLMブロックを併用のケースを実験
- ESP32, Raspberry Pi Picoはソフトウェア実装を使用
- Tang Nano 9K
  - 演算回路をRTLで実装,汎用PCとUART(FTDIのチップ)を介してデータ通信
  - UART IPを使用し、ボーレートは正しく動作した最大の921600に設定
  - UARTのインターフェースと演算回路を別のクロックドメインにし、境界(CDC)は4-phase handshake

処理時間の比較

- 最大動作周波数におけるレイテンシを計測

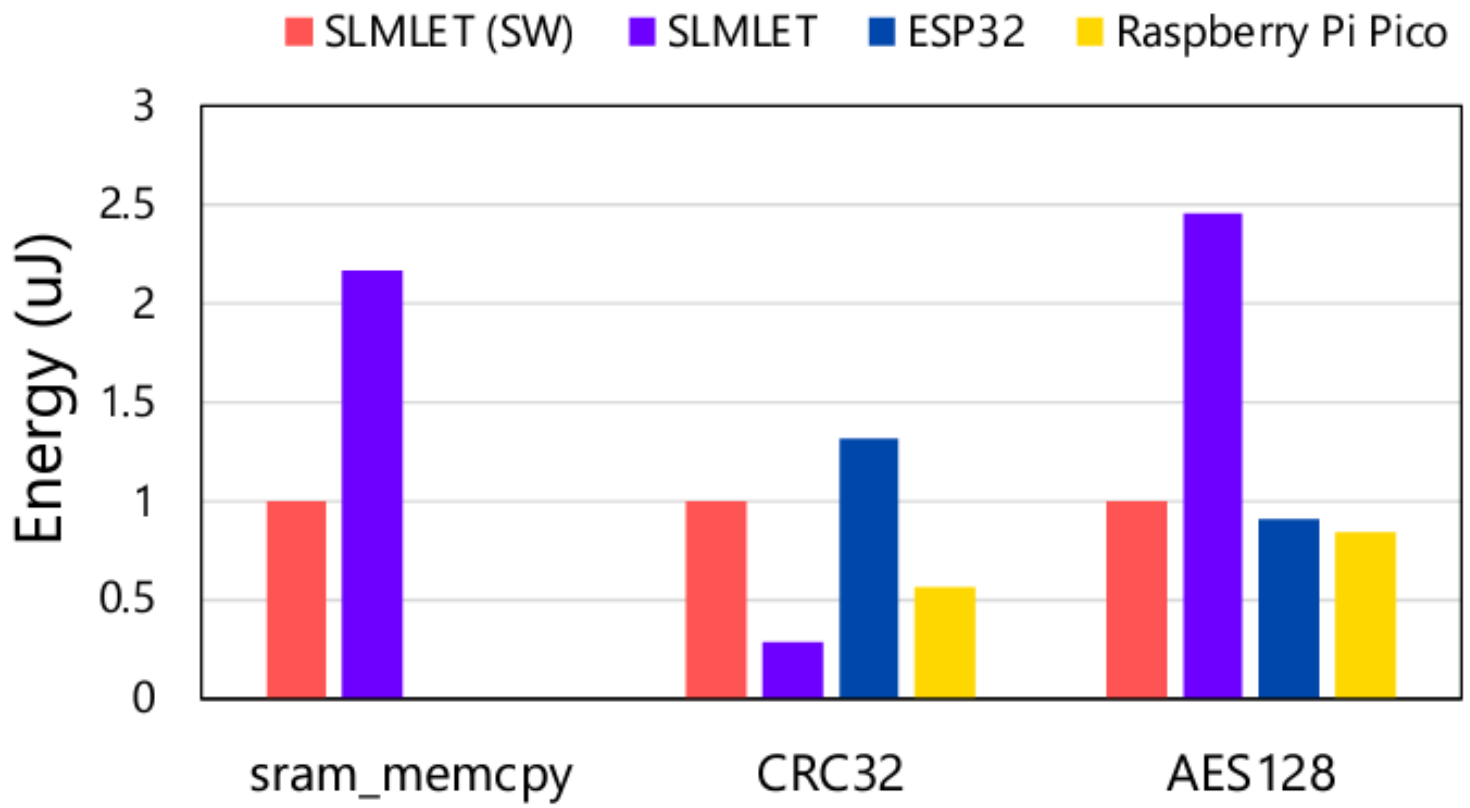
時間: us, 周波数: MHz

	sram_memcpy		CRC32		AES128	
	時間	周波数	時間	周波数	時間	周波数
SLMLET (SW)	95.60 (28,679)	300	47.05 (14,117)	300	6.217 (1,865)	300
SLMLET	329.9 (32,994)	100	30.73 (2,151)	70	34.86 (976)	28
ESP32	n/a	n/a	77.58	240	7.814	240
Raspberry Pi Pico	n/a	n/a	66.59	200	14.00	200
Tang Nano 9K	n/a	n/a	885.7	50	1333	34

()書きはサイクル数

1ワードのCRC32を計算を1サイクルで行うハードウェアを実装、サイクル数を大きく削減
周波数はソフトウェアより低いがそれでも時間を短縮
Tang Nano 9KはUART通信が支配的

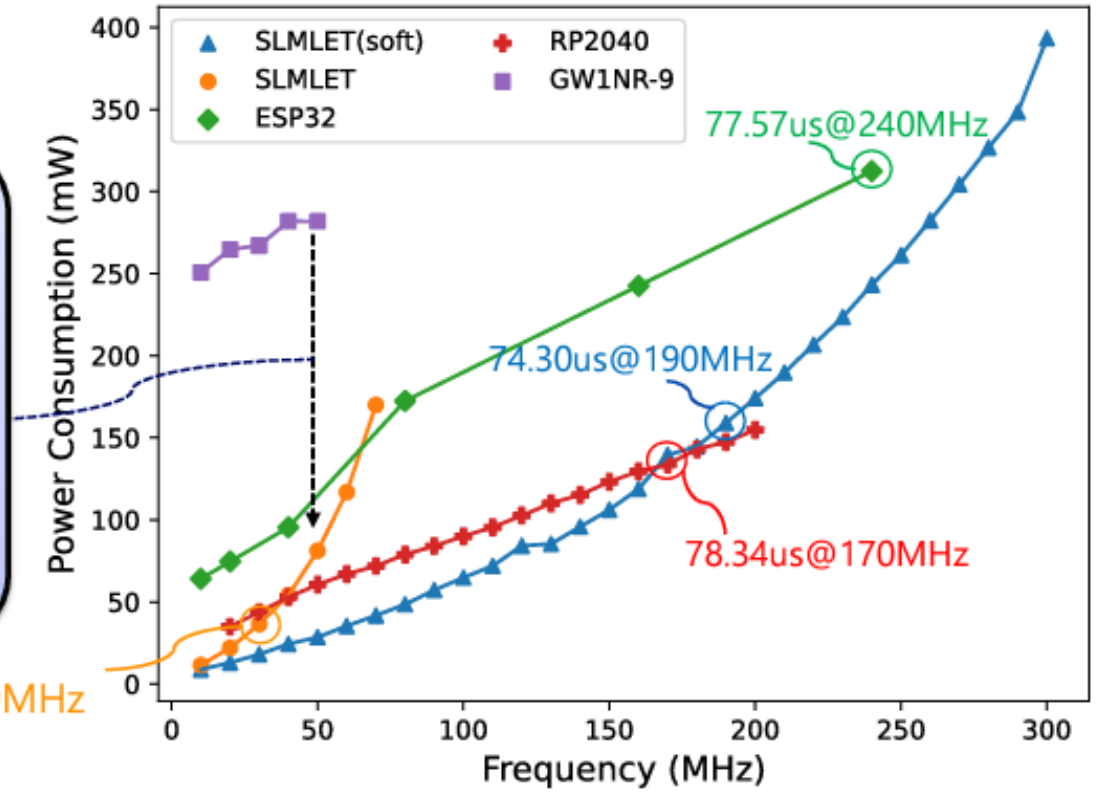
消費エネルギーの比較（最高性能時）



- SLMLET (SW実装)のケースで正規化
- CRC32ではSLMブロックを使うことで以下のエネルギー削減を達成
  - SLMLET (SW)に対し 71%, ESP32に対し 78%, Raspberry Pi Picoに対し 49%

消費電力の比較

FPGA上の演算回路はTang Nano 9Kは50MHz
SLMブロックは70MHz
→ 50MHzではリバースバイアスによるリーク削減が可能
VPW=-0.3 Vで**5mW**程度のリーク電力



- 70us程度の処理時間となる動作点で比較した時の電力削減
  - SLMLET (SW)に対し 77%, ESP32に対し 88%, Raspberry Pi Picoに対し 72%

SLMLET評価のまとめ

- 全機能が一通り動作することを確認
- SLMLETのRISC-VとSLMの協調開発フロー、ライブラリを実装
- 実験により以下がわかった
  - 再構成処理の並列化、ビットストリームの圧縮を行うことで先行研究と比べても十分に高速な回路切り替えが可能
  - 単純な回路であれば100MHz程度の動作が可能
  - ブロック2の再構成コントローラ部で未解決の不具合が発見
- エッジ向けの他のデバイスと比較
  - CRC32のようなハードウェア化の恩恵が大きいベンチマークではレイテンシの短縮とエネルギーの削減を確認
- SLMLET-2の開発は以下の原因で遅れている
  - USJCの代理店がTEDからToppanに変更
  - PDKの消去と再配布を行うことになった
  - しかしNDA締結に時間が掛かりまだ締結されていない
- 来年3月のランに間に合わせたいがわからない

2022年後半-2023年前半の業績

https://www.am.ics.keio.ac.jp/crest/?page_id=37

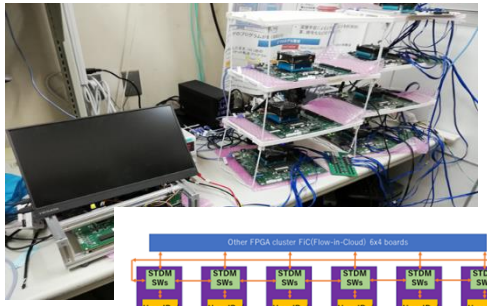
- ジャーナル件数：15件　IEEE Trans., IEEE Microなど一流ジャーナルもあり
- 国際会議：20件　FPL、ICFPTなど難関国際会議も含む
- 受賞：8件
- イベント
  - FIT2023：2023年9月8日　「Society 5.0社会のためのコンピューティング技術をどう生かすか？」
  - 応用物理学会 2023年3月16日　半導体工場のIT化　「MEC用FPGAクラスタによる工場制御」
  - RVfpga Workshop　2023年3月7，8日　慶應義塾大学矢上キャンパス
  - ロボット聴覚オープンソースソフトウェアHARK講習会，2023年11月23日, 慶應義塾大学矢上キャンパス
- 展示、デモ：
  - 2022年12月　慶應テクノモールで成果を展示
  - 2023年5月　国際学会FCCMで成果をデモ
  - 2023年6月　RISC-V dayでSLMLETのデモ

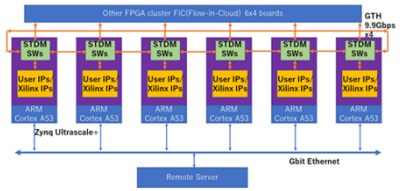
現状

プライベート情報のエッジ
地域カプセル化を支える
ハードウェアとアプリ設計

System-C・手動分割
自動インタフェース挿入、
全体シミュレーション

Slurm/MKUBOSWWW/
MKUBOSMNR/PYNQ





MKUBOS クラスタ

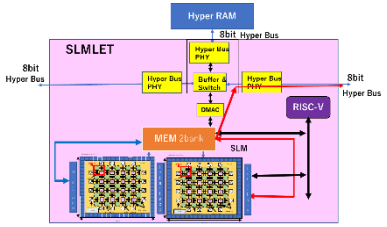
課題終了時の成果イメージ

個人情報保護やシステム永続
化に寄与する地域サービス
アーキテクチャの確立

System-C/C++・自動分割
自動インタフェース挿入、
全体シミュレーション

MEC-RM（ミドルウェア）
ヘテロジニアスマルチクラスタ管理

MKUBOS HPFクラスタ



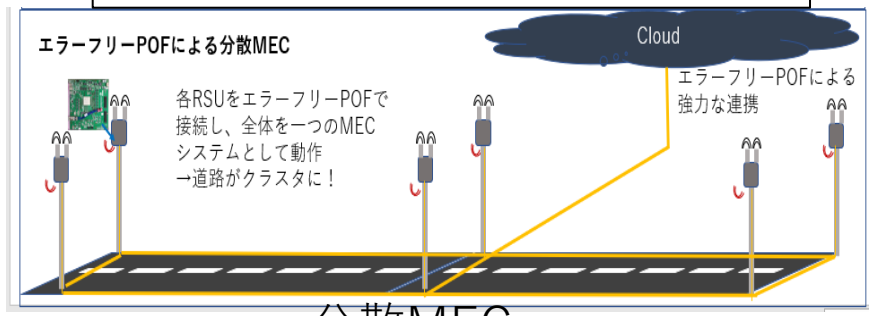
SLMLET クラスタ

その後の展開

・2つの関連一社代表として実展開と事業化
・IEEE-SA関連WGのチェアとして標準化

System-C/C++・自動分割
自動インタフェース挿入、
全体シミュレーション

高速大容量バイナリ通信支援、
OSS公開



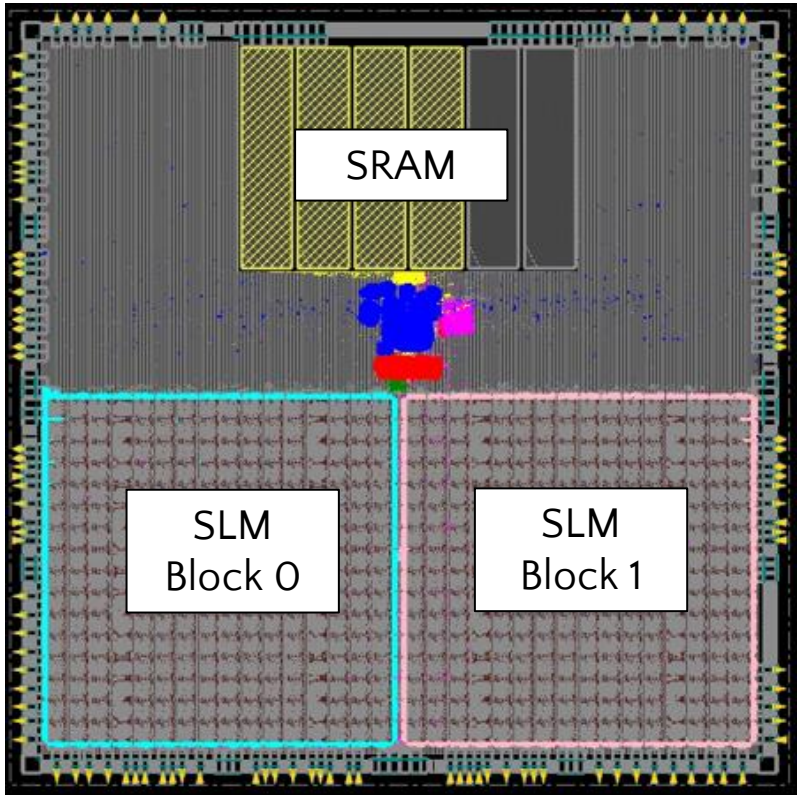
分散MEC

NVFFの導入
IP化、国産FPGAへ

Crust-Core型チップ
Agile-Xとの連携

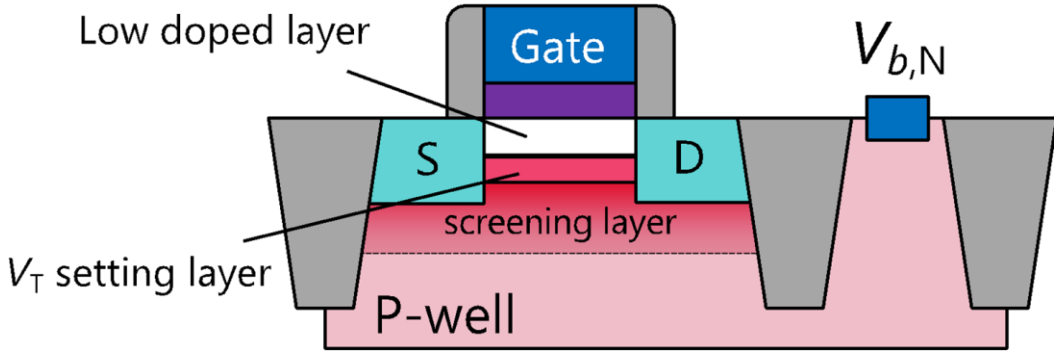
質問用

SLMLETのプロトタイプ実装



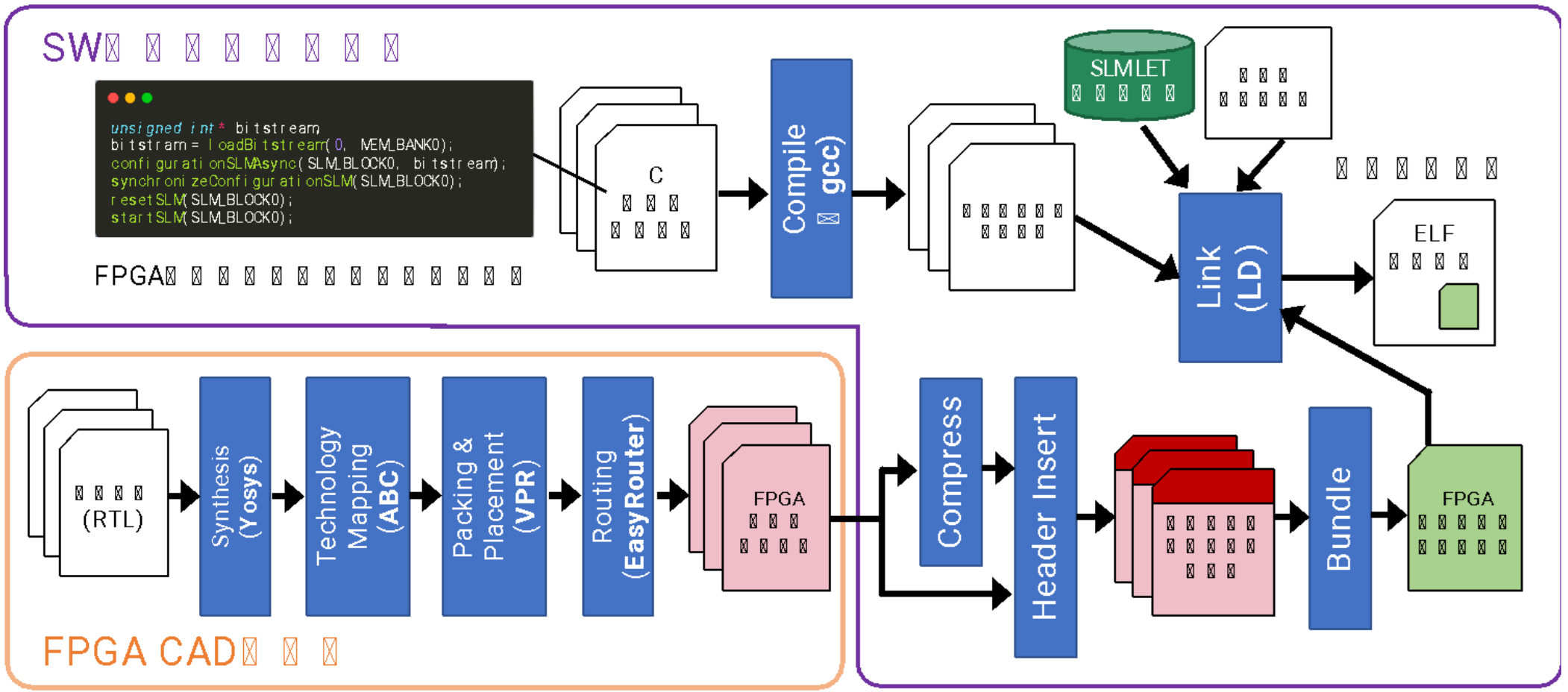
チップレイアウト 4.2mm角

- USCJ 55nmを採用
  - 閾値電圧のばらつきが小さい
  - 240mV/V の高い基板効果係数
    → ボディバイアス制御によるリーク削減の期待
- スタンダードセル: C55DDCT07L60LVT



USCJプロセスのトランジスタ構造

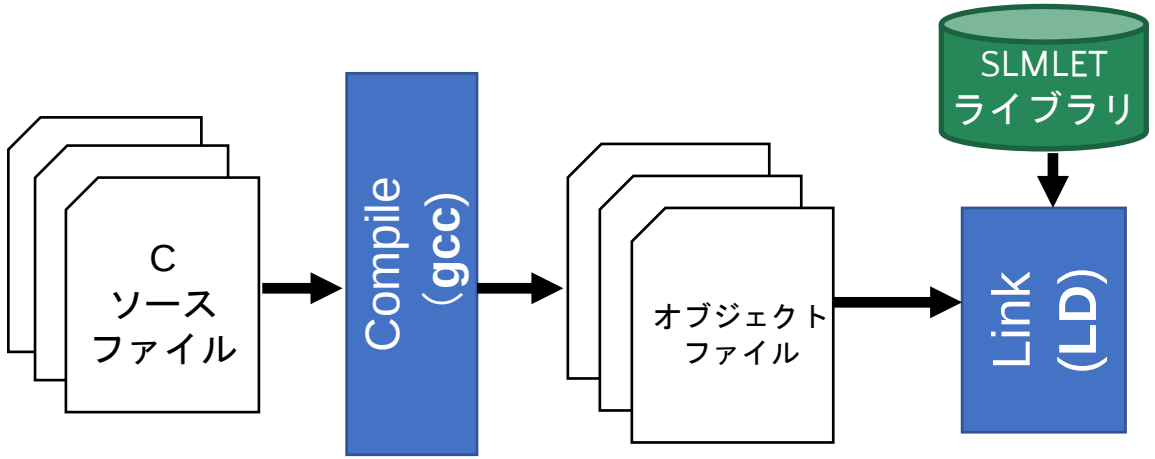
HW/SW設計フロー：全体像



HW/SW設計フロー: SW記述

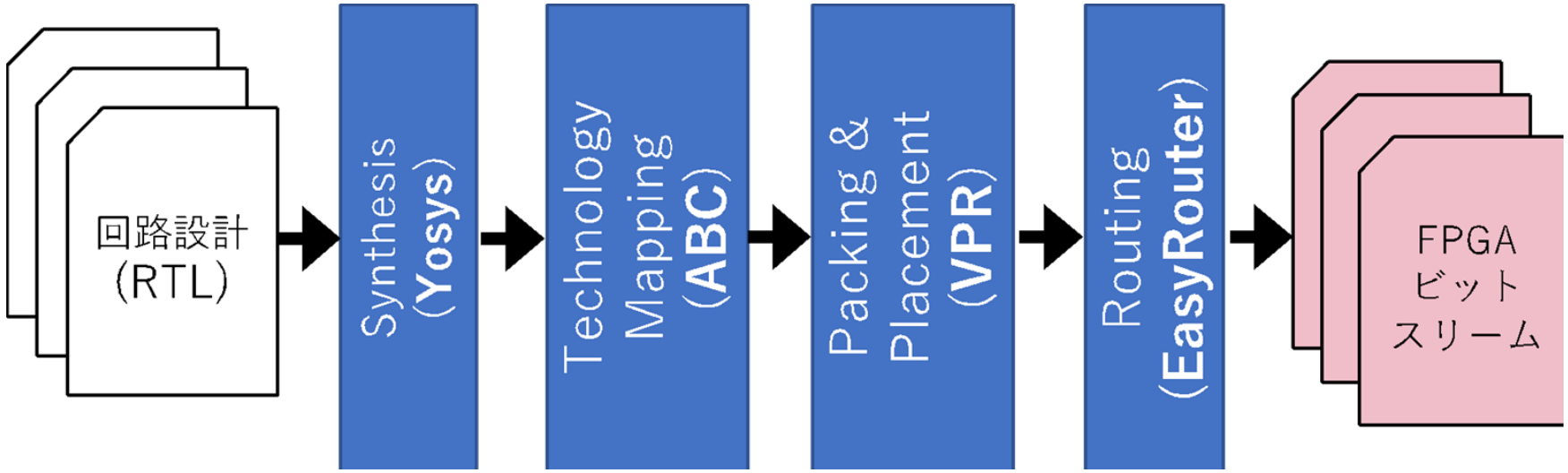
```
unsigned int* bitstream;
bitstram = loadBitstream(0, MEM_BANK0);
configurationSLMAsync(SLM_BLOCK0, bitstream);
synchronizeConfigurationSLM(SLM_BLOCK0);
resetSLM(SLM_BLOCK0);
startSLM(SLM_BLOCK0);
```

FPGAコンフィギュレーションコード



- RISC-Vコアで実行されるコードはC言語で記述
- SLMブロック制御
  - ライブラリが提供する関数(後述)を呼び出す

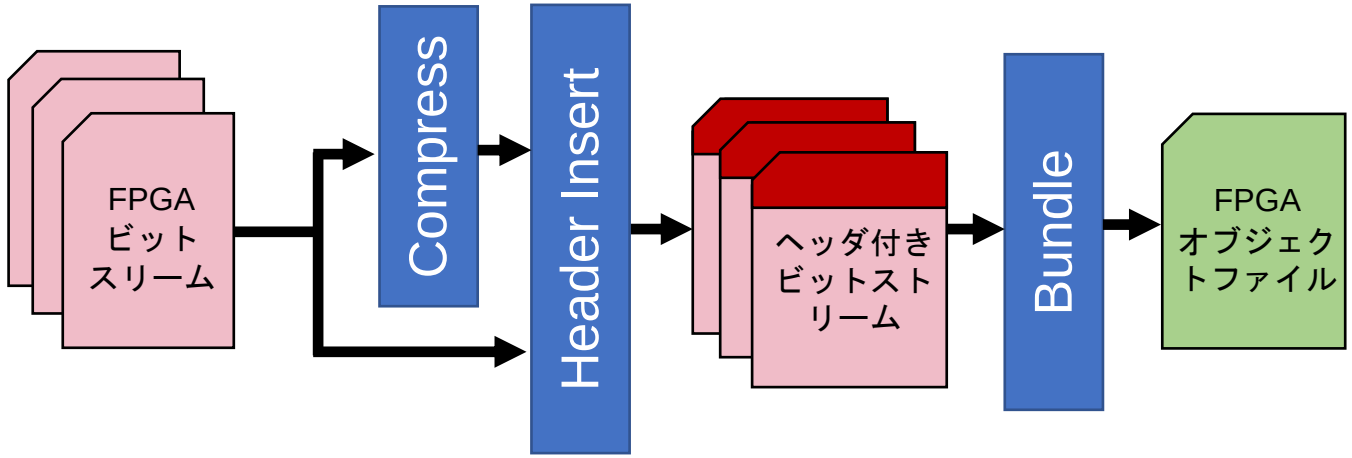
HW/SW設計フロー: HW設計



- FPGA部に構成するハードウェアの設計
  - 先行研究[1]で整備されたFPGA CADを使用
  - OpenFPGAなど広く使用されるYosys, ABC, VPRにEasyRouter[2]を組み合わせ

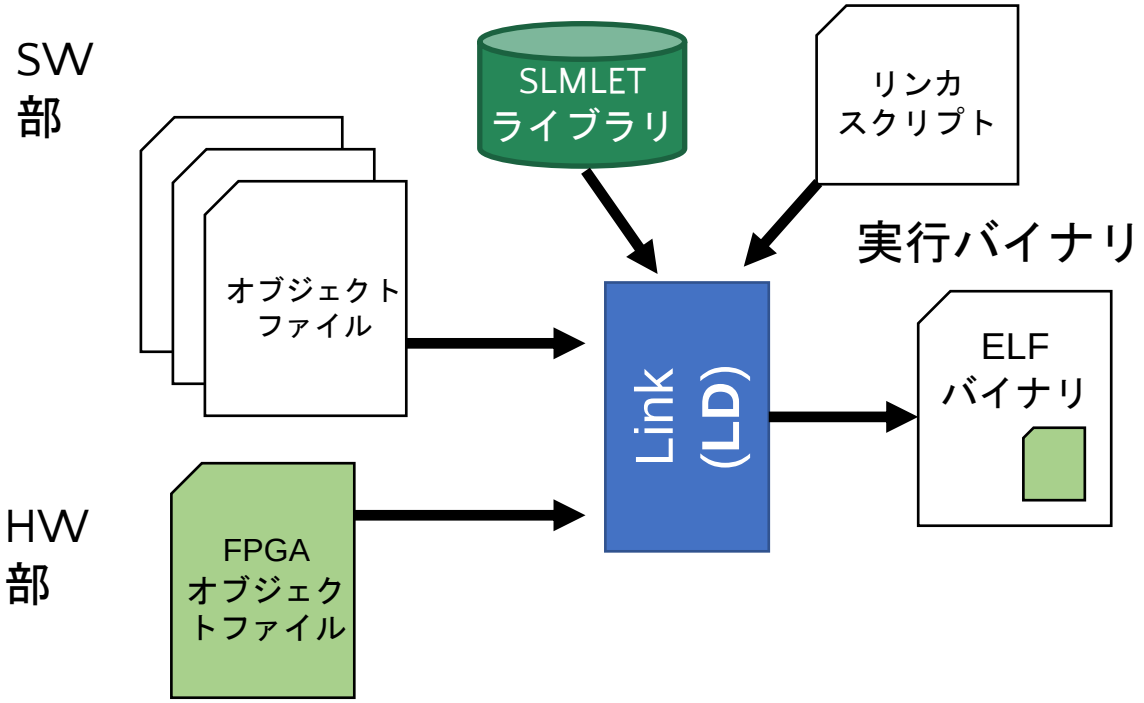
[2] Zhao, Qian, et al. "An automatic FPGA design and implementation framework." 2013 23rd International Conference on Field programmable Logic and Applications. IEEE, 2013.

HW/SW設計フロー: ビットストリームのオブジェクト化



- ビットストリームに対する前処理
  - Tag-Less Compressionによる圧縮 (オプション)
  - ヘッダー挿入
- バンドル化
  - 複数のビットストリームを一つのオブジェクトファイルにまとめる

HW/SW設計フロー: 実行ファイルの生成



- リンカによって実行バイナリを生成
- カスタムなリンカスクリプト
  - 命令メモリとデータメモリの分離
  - メモリマップの記述
  - ライブラリのためのシンボル埋め込み

プログラム例

1. コンフィギュレーションデータを共有メモリ領域に読み出し

```
printf("load bitstream\n");
unsigned int* bitstream = loadBitstream(0, USE_SRAM_BANK);
if (!bitstream) {
    printf("failed to load bitstream\n");
    return 1;
}
```

2. コンフィギュレーションデータをSLMに書き込み(再構成)

```
printf("start configuration\n");
configurationSLMAsync(SLM_BLOCK0, bitstream);
// do something
synchronizeConfigurationSLM(SLM_BLOCK0);

printf("enable SLM\n");
resetSLM(USE_SLM_BLOCK);
startSLM(USE_SLM_BLOCK);
```

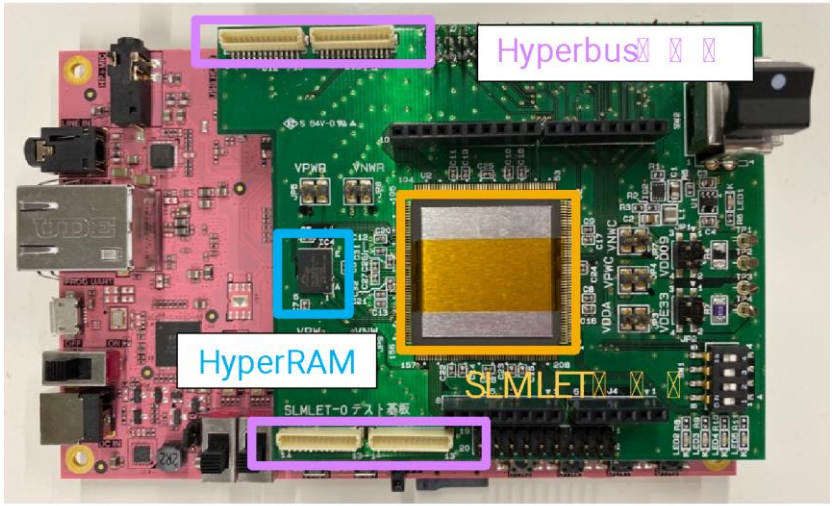
3. リセット、開始信号の送信

4.SLM Memory mapped領域への書き込み

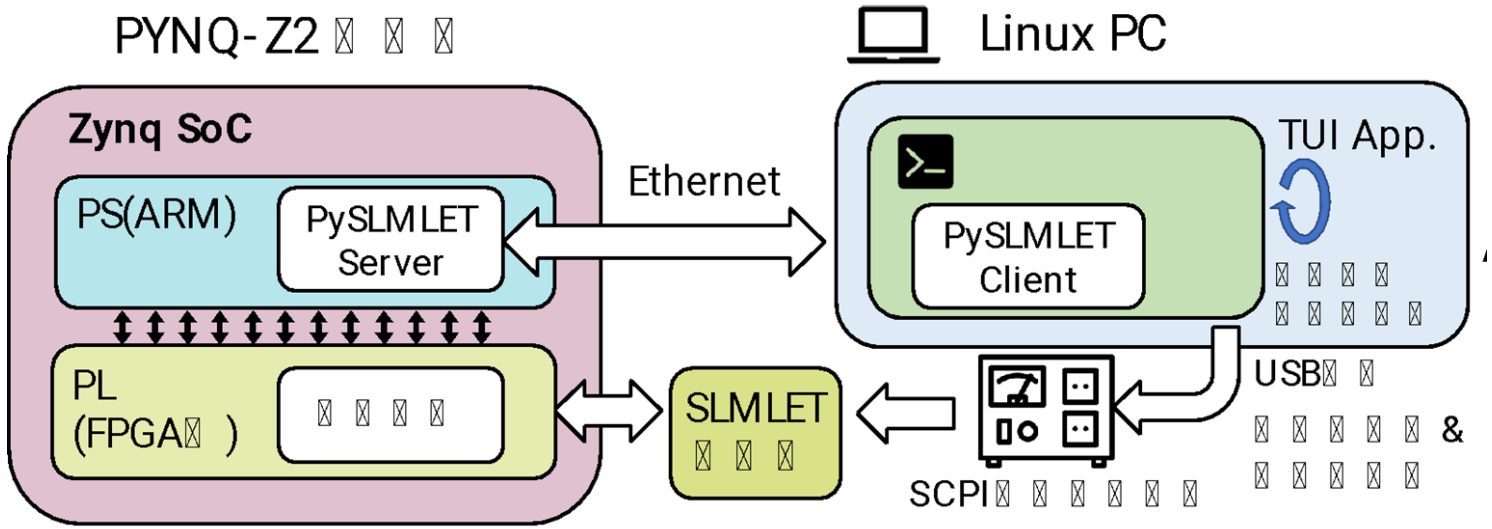
```
printf("write data to SLM\n");
for (i = 0; i < FPGA_REG_COUNT; i++) {
    writeSLM(USE_SLM_BLOCK, (int*)(4 * i), &write_data[i]);
}

printf("read data from SLM:\n");
for (i = 0; i < FPGA_REG_COUNT; i++) {
    printf("FPGA Reg %d: %08X\n", i, readSLM(USE_SLM_BLOCK, (int*)(4 * i)));
}
```

5.SLM Memory mapped領域からの読み出し

評価環境: PySLMLET-TUI



SLMLETボードとPYNQ-Z2



- PySLMLET-TUI
  - 実行バイナリや電圧、周波数など条件を変更しテストを行うText-based UI
  - 簡単なシェルスクリプトで自動化測定を可能に
  - 先行研究でGUI版は実装済み [デモ動画]
- PySLMLET[3]
  - PYNQ FPGAはSLMLETのホストとして振る舞う回路が実装
  - Pythonのドライバでこれらを制御

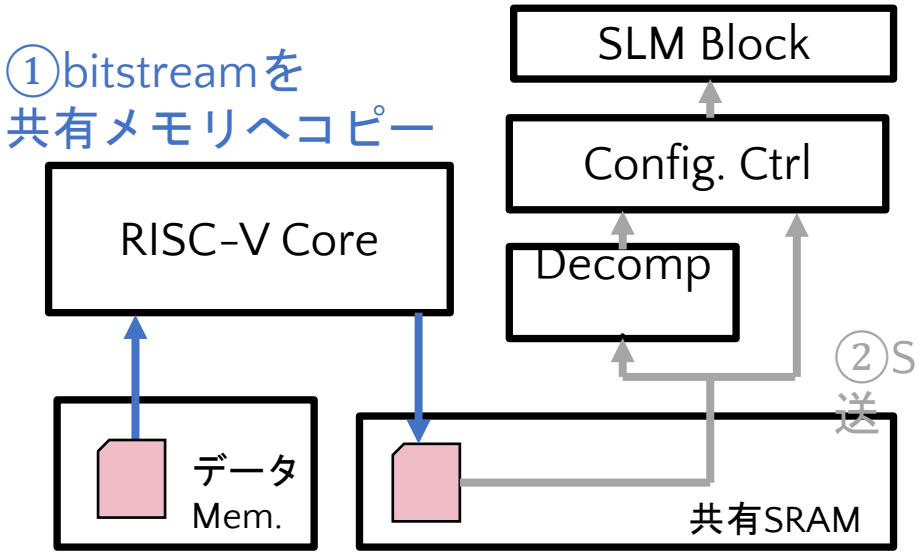
評価環境の概要

- pynqモジュールのimportに時間がかかりすぎる (20sec程度)
- サーバとクライアントに分離
- TUIはTCP/IPでサーバに接続
- TUI appの起動時間を短縮
- TUIを別マシンでも実行可能に

[3] 小島拓也, et al. "Jupyter Notebook を介した RISC-V SoC 向け実機テスト環境の構築." 研究報告組込みシステム (EMB) 2023.24 (2023): 1-7.

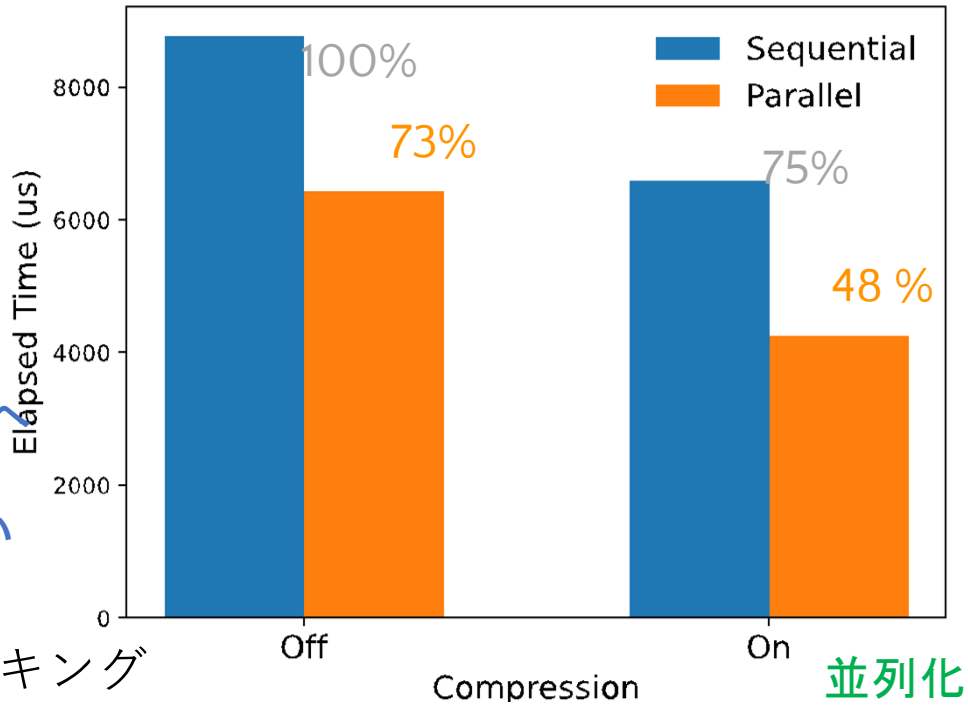
実験と評価

評価: 再構成にかかる時間



圧縮してもコントローラがSLMブロックに書き込む時間は同じ

並列化してもSRAMへのコピーは直列
→ 半分にはならない

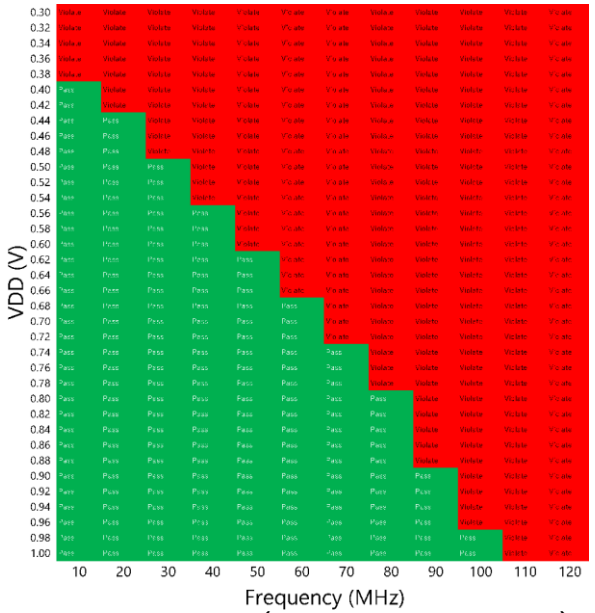


並列化&圧縮によりおよそ半分の時間に

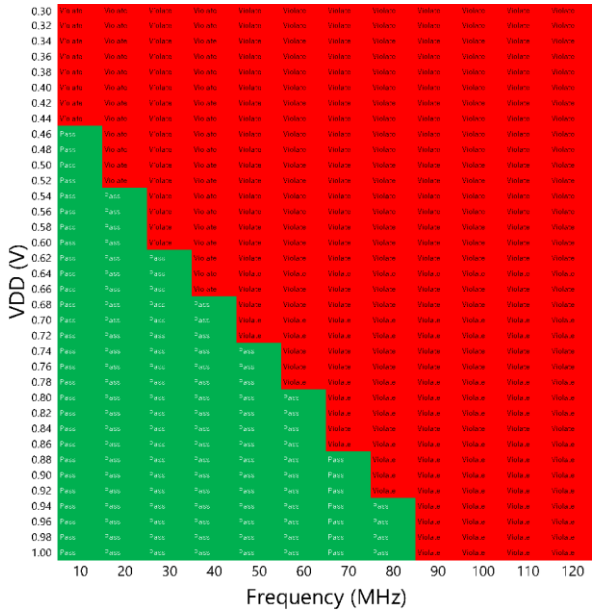
- 評価パターン
  - 二つのブロックを直列 or 並列に再構成
  - 再構成のライブラリ関数 ブロッキング or ノンブロッキング
  - TLCによる圧縮 or 非圧縮
- FPGA上の回路: RISC-Vから読み書き可能なメモリマップトなレジスタ 32bitx4本
  - SLM使用率: 29%, データ圧縮率49%
- 最新の類似研究[4]では6720 6-LUTの再構成に1msだがプロセッサは最大1.1GHz程度で動作
  → SLMLETの再構成時間は十分に高速といえる

[4] Chang, Ting-Jung, et al. "CIFER: A 12nm, 16mm 2, 22-Core SoC with a 1541 LUT6/mm 2 1.92 MOPS/LUT, Fully Synthesizable, CacheCoherent, Embedded FPGA." *2023 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2023.

実機評価: SLMブロックの動作範囲

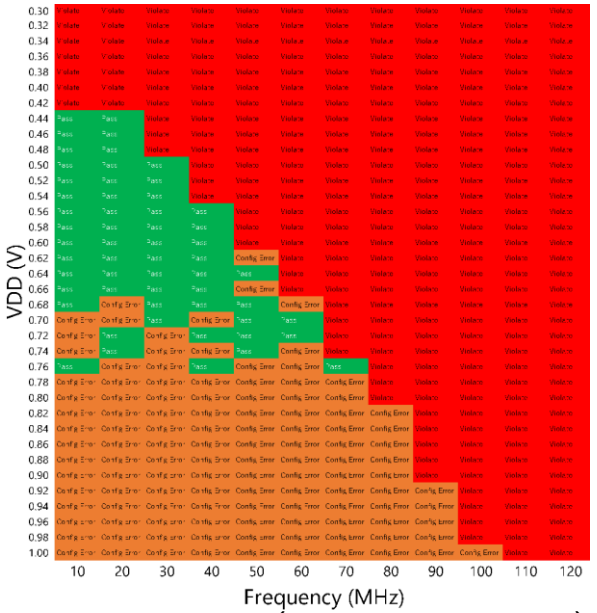


VPW=0 V (f_{max} = 100MHz)

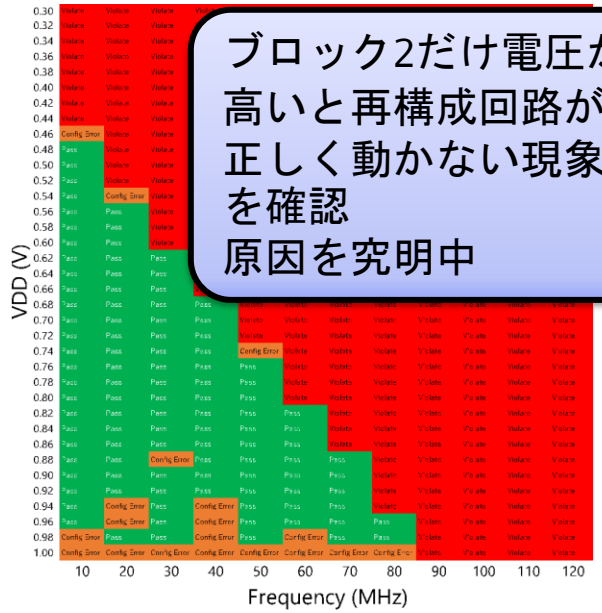


VPW=−0.4 V (f_{max} = 80MHz)

ブロック1



VPW=0 V (f_{max} = 100MHz)



VPW=−0.4 V (f_{max} = 80MHz)

ブロック2

ブロック2だけ電圧が高いと再構成回路が正しく動かない現象を確認
原因を究明中

- 2つのブロックをそれぞれ正しく再構成および計算ができる電圧、周波数を測定
  - 先ほど同じメモリマップトレジスタの回路を使用 (最大SLM段数 6)
  - 電圧: 0.30–1.00 V (標準 0.90 V), 0.02 V刻み
  - 周波数: 10–120MHz, 10MHz刻み
  - ボディバイアス電圧: 0.0 V, −0.1 V, −0.2 V, −0.3 V, −0.4 V
- 動作特性は概ね同じ、想定よりも高い周波数でも動作した

失敗

失敗

再構成だけ失敗

他のシステムと比較

	SLMLET (本研究)	ESP32	Raspberry Pi Pico	Tang Nano 9K
実装プロセス	USJC 55nm	TSMC 40nm	40nm	TSMC 55nm
CPU	RV32I	Xtensa LX6	ARM Cortex-M0+	N/A*
定格クロック	N/A	240MHz	133MHz	N/A
FPGA	5-SLM 896 x2	N/A	N/A	4-LUT 8640
使用した開発環境	SLMLET SDK	ESP IDF 5.0.2	Pico SDK 1.5.1	Gowin 1.9.8.11 IDE
GCC	12.2.0	11.2.0	10.3.1	N/A
備考	チップ電力を測定	ボード電力を測定 通信モジュールはプログラムから明示的にOFF	ボード電力を測定 200MHzまでオーバークロックで動作を確認	ボード電力を測定 FTDI D2XX Driver 1.4.27

*ボード上に別チップとしてBL702(RISC-V,144MHz)が搭載しているが基本はUART,JTAG用

- SLMLETはRISC-Vコアのみ&SLMブロックを併用のケースを実験
- ESP32, Raspberry Pi Picoはソフトウェア実装を使用
- Tang Nano 9K
  - 演算回路をRTLで実装,汎用PCとUART(FTDIのチップ)を介してデータ通信
  - UART IPを使用し、ボーレートは正しく動作した最大の921600に設定
  - UARTのインターフェースと演算回路を別のクロックドメインにし、境界(CDC)は4-phase handshake

比較実験: ベンチマーク

- ベンチマーク
  1. sram_memcpy (SLMLETのみ)
     - 共有SRAM上のデータ16KBをコピー
     - SLMブロックのSRAMインターフェースを使う回路
  2. CRC32
     - 1KBのバイナリデータからCRC32を計算(生成多項式0x04C11DB7)
     - ソフトウェア実装はMiBenchの実装を使用
  3. AES128
     - 128bit長の鍵を用いて1ブロックのAES暗号化を行う
     - ソフトウェア実装はMiBenchの実装を使用
- いずれもO3でコンパイル

結果: 処理時間の比較

- 最大動作周波数におけるレイテンシを計測

時間: us, 周波数: MHz

	時間	**周波数**	**時間**	**周波数**	**時間**	**周波数**
SLMLET (SW)	95.60 (28,679)	300	47.05 (14,117)	300	6.217 (1,865)	300
SLMLET	329.9 (32,994)	100	30.73 (2,151)	70	34.86 (976)	28
ESP32	n/a	n/a	77.58	240	7.814	240
Raspberry Pi Pico	n/a	n/a	66.59	200	14.00	200
Tang Nano 9K	n/a	n/a	885.7	50	1333	34

()書きはサイクル数

SLMブロックはSRAMインターフェースが16bitなので
ハードウェア化してもソフトウェア実装を上回ることができない

結果: 処理時間の比較

- 最大動作周波数におけるレイテンシを計測

時間: us, 周波数: MHz

	時間	**周波数**	**時間**	**周波数**	**時間**	**周波数**
SLMLET (SW)	**95.60** (**28,679**)	300	47.05 (14,117)	300	**6.217** (1,865)	300
SLMLET	329.9 (32,994)	100	**30.73** (**2,151**)	70	34.86 (**976**)	28
ESP32	n/a	n/a	77.58	240	7.814	240
Raspberry Pi Pico	n/a	n/a	66.59	200	14.00	200
Tang Nano 9K	n/a	n/a	885.7	50	1333	34

()書きはサイクル数

1ワードのCRC32を計算を1サイクルで行うハードウェアを実装、サイクル数を大きく削減
周波数はソフトウェアより低いがそれでも時間を短縮
Tang Nano 9KはUART通信が支配的

結果: 処理時間の比較

- 最大動作周波数におけるレイテンシを計測

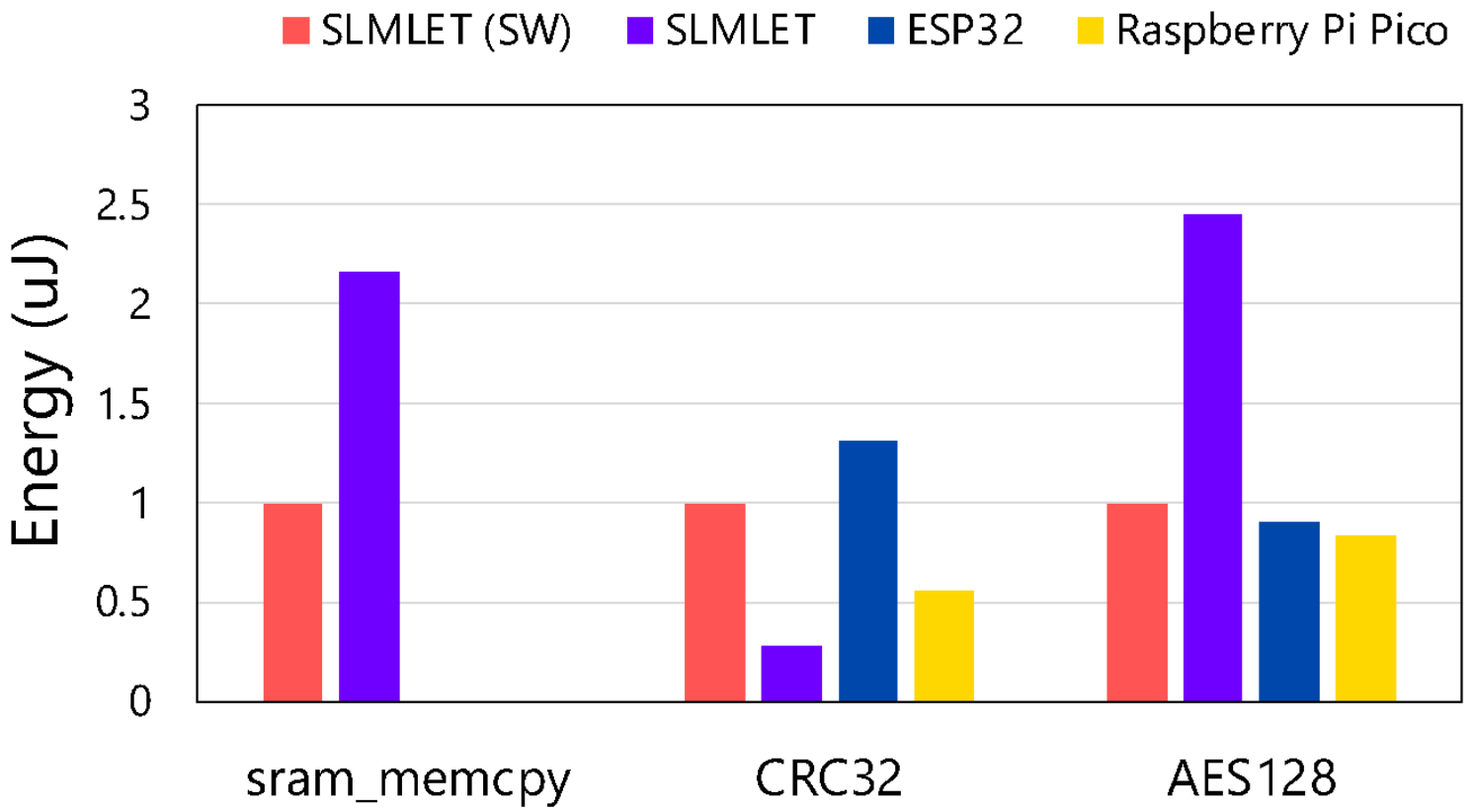
時間: us, 周波数: MHz

	時間	**周波数**	**時間**	**周波数**	**時間**	**周波数**
SLMLET (SW)	95.60 (28,679)	300	47.05 (14,117)	300	6.217 (1,865)	300
SLMLET	329.9 (32,994)	100	30.73 (2,151)	70	34.86 (976)	28
ESP32	n/a	n/a	77.58	240	7.814	240
Raspberry Pi Pico	n/a	n/a	66.59	200	14.00	200
Tang Nano 9K	n/a	n/a	885.7	50	1333	34

()書きはサイクル数

SLMブロックを2つ使用し、専用回路を実装
サイクル数は減少したが、二つのブロックにまたがる
クリティカルパスによって動作周波数が低い
結果的にはソフトウェア実装の方が早い

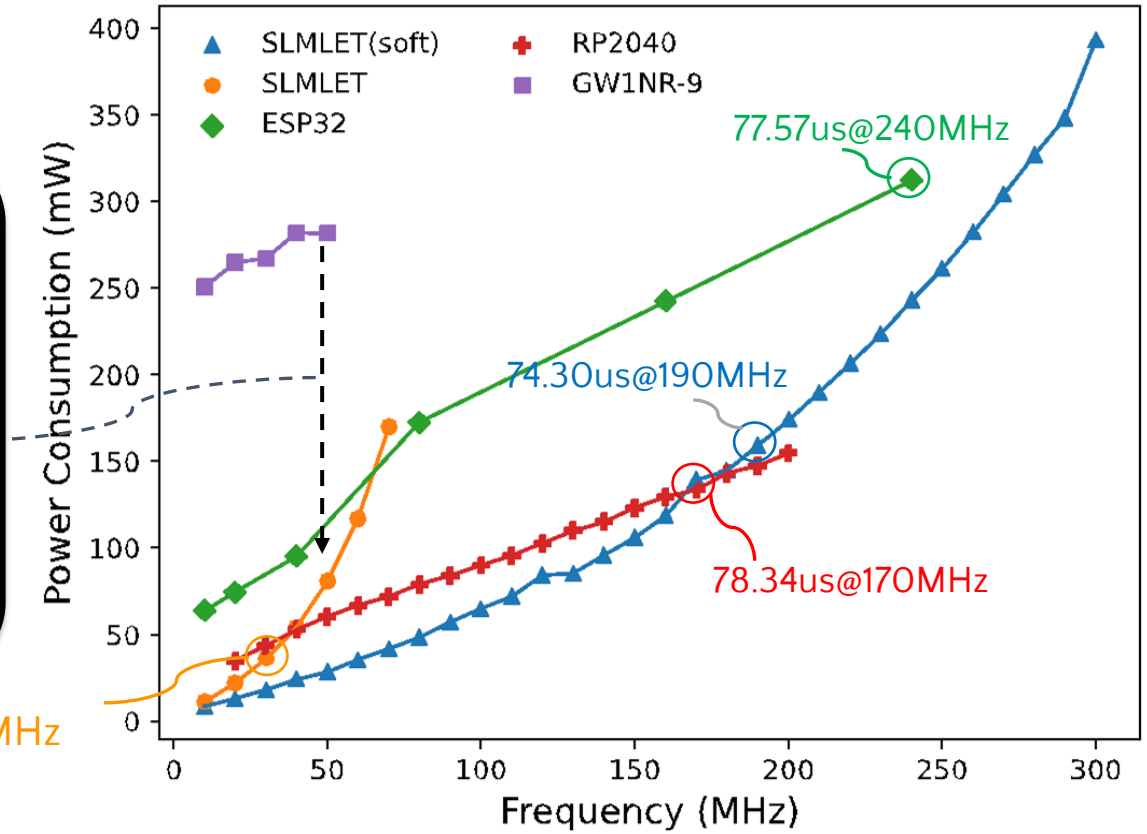
消費エネルギーの比較 (最高性能時)



- SLMLET (SW実装)のケースで正規化
- CRC32ではSLMブロックを使うことで以下のエネルギー削減を達成
  - SLMLET (SW)に対し 71%, ESP32に対し 78%, Raspberry Pi Picoに対し 49%

消費電力の比較 (CRC32)

FPGA上の演算回路はTang Nano 9Kは50MHz
SLMブロックは70MHz
→ 50MHzではリバースバイアスによるリーク削減が可能
VPW=-0.3 Vで**5mW**程度のリーク電力



- 70us程度の処理時間となる動作点で比較した時の電力削減
  - SLMLET (SW)に対し 77%, ESP32に対し 88%, Raspberry Pi Picoに対し 72%

TEG3への課題

- TEG2実チップ評価はまだですが、設計段階でわかったこと
  - 複数のIPを用いることは善し悪し

【メリット】

1. デバイス設計が楽。ならべるだけ
2. 実装する回路を工夫すれば、IP間のマイグレーション（新機能）が可能
3. EDAツールの変更が不要

【デメリット】

1. デバイス設計が楽だが、回路実装は複雑になりデメリットも多い
2. 2個以上になるとIP間の接続関係が複雑化する
3. さらに、大規模な回路の分割という余分な作業も発生
4. I/Oの自由度が高すぎて扱いが面倒

メリットよりデメリットが多いのでTEG3では設計方針を見直します。

MEC用ASIC　SLMLET

- 新しい組み込みFPGA　SLMを搭載
- HyperBusによるネットワーク機能
- RISC-Vによる制御

-> 従来のPCの100倍以上エネルギ効率を改善

- ダイサイズ：4.2mm□
- 半導体プロセス：USJC 55nm LVT