オープンソースな
ROS2通信ハードウェアIP
“ROS2rapper”

https://github.com/AXE-jp/ros2rapper
↑にて絶賛公開中

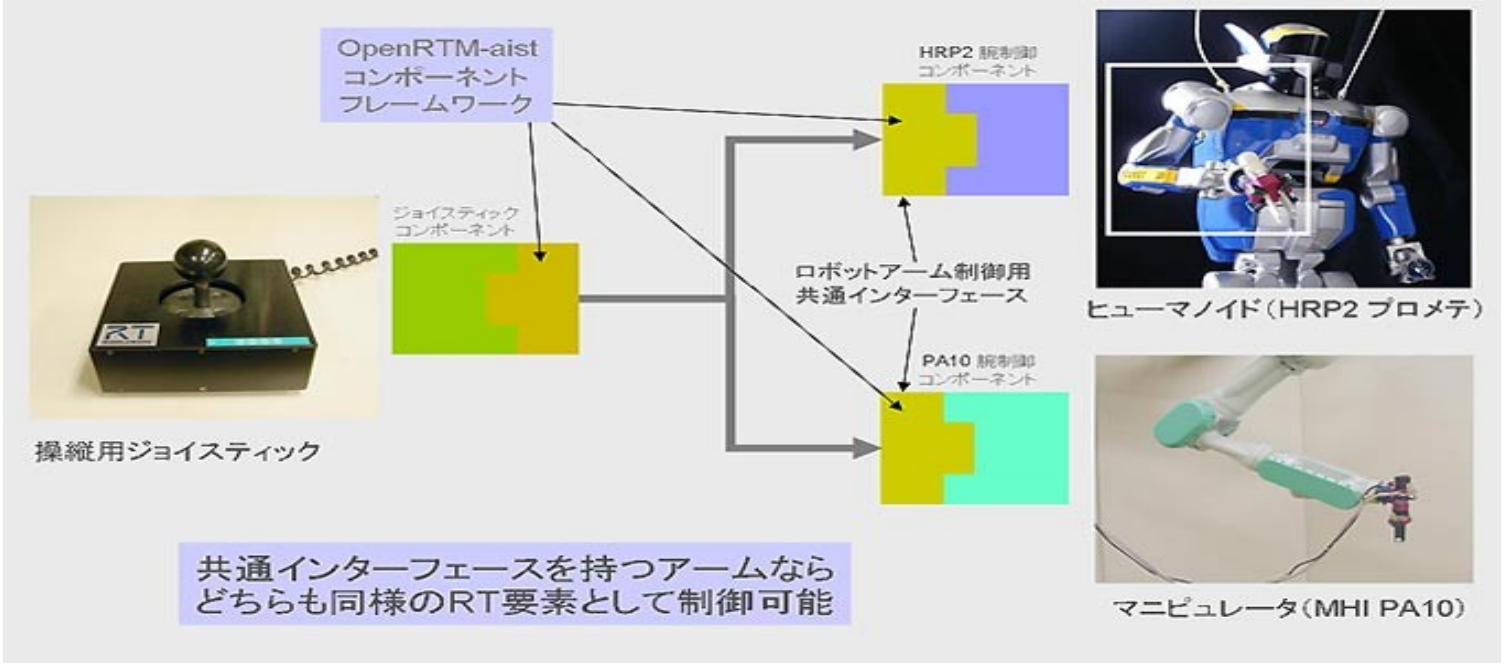
2025/JUN/25
AXE, Inc.

ROS2は”ソフトウェア・バス”

ROS ロボット・ミドルウェア = ソフトウェア・バス

ロボットのモジュールの流通性が高まる

日本のロボット業界は、ROSをデファクト・スタンダードにしようと活動している



引用元 http://www.aist.go.jp/aist_j/press_release/pr2005/pr20050224/pr20050224.html

ROS2って、Linuxが要るんじゃね?

- “ros2rapper”は、No OSだ!
- コンパクト
- 高速
- 堅牢
  - ハードウェア論理なので、ウイルスがつかない

“ROS2rapper”

https://github.com/AXE-jp/ros2rapper

ROS2プロトコルを完全ハードウェア化

AXEでは、ROS2プロトコルを完全ハードウェア化した”ROS2rapper”

- ソフトウェア技術者がハードウェア論理を書き、LSI化。現実に!
- OSSとして、近日 配布開始
  - GPLと AXEプロプライエタリの、ダブル・ライセンスを検討中
- CPU無し,Linux無しで、ロボットの部品モジュールができる
  - センサとROS2プロトコルHWだけで、センサ・モジュール
  - PWMとROS2プロトコルHWだけで、アクチュエータ・モジュール
  - アプリケーションはC(HLS)言語で書いておけば、すぐハードウェア論理に合成
- ロボット部品が、ゴミのようなLSIでできる ← CPU不要
- CPU脳の敗北
- ハードウェアなので、堅牢 かつ 高速。そしてコンパクト

※ROS2ハードウェアには、コンフィギュレーション用のPROMがあることが望ましい

Arty A7-35ボード(xc7a35ti-csg324-1Lチップ)において

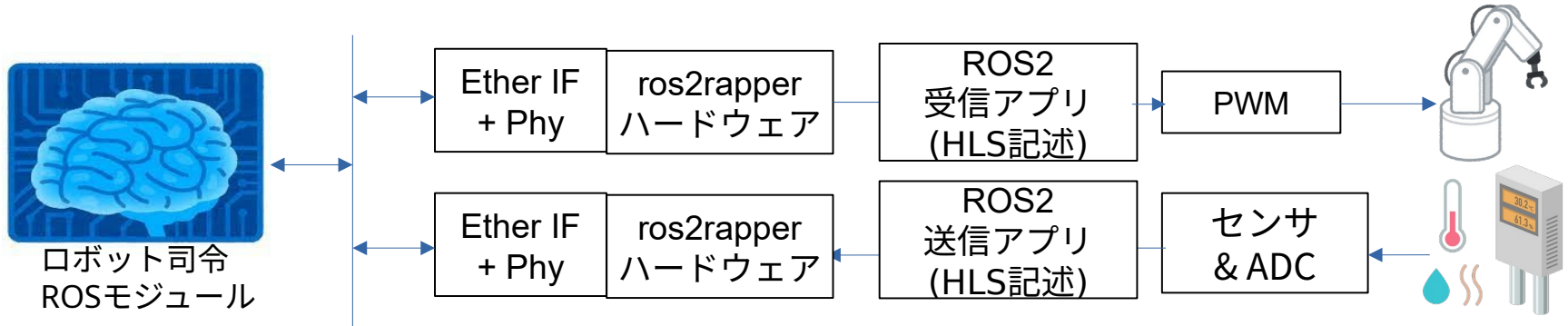
FPGA使用資源

- LUT: 33666
- FF: 13087

最大周波数: 121.01MHz

送信パケット生成 所要時間:
- IPデータグラム生成複数サイクル版:
  - 14サイクル=140nsec@100MHz
- IPデータグラム生成1サイクル板):
  - 8サイクル=160nsec@50MHz

受信 処理時間:
- 46サイクル=460nsec@100MHz



実現している機能

Table 1.1 サポートしている機能の一覧

機能		内容	備考
プロトコル	RTPS (SPDP /SEDP)	ROS2トピックのパブリッシュ	
		ROS2トピックのサブスクライブ	
	UDP/IP	UDPデータグラムの送信	
		UDPデータグラムの受信	
通信物理層	Ethernet	Ethernet層	Ethernet以外の物理層に差し替え可能
	ARP	物理アドレス解決	

- 各機能は、個別に有効化/無効化できる
  - ROS2トピックのパブリッシュ
  - ROS2トピックのサブスクライブ

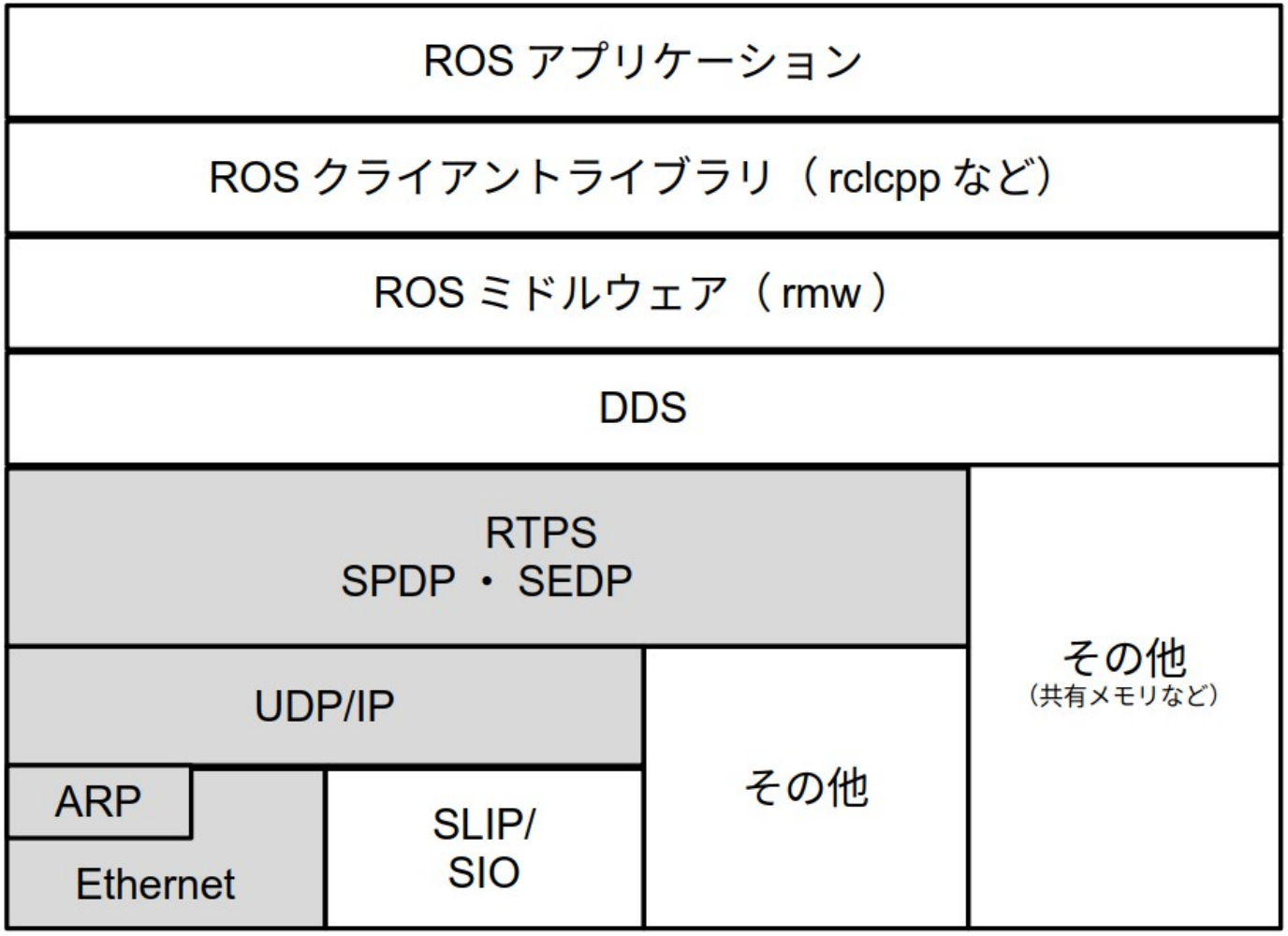


Fig.1.1 ROS2 アーキテクチャにおける ROS2rapper の位置

ROS2プロトコル

- RTPS
  - 低位プロトコル
- SPDP
  - 参加者 情報を送信
    - 信頼性が無い。マルチキャスト
  - 一定時間ごとに、マルチキャスト(ブロードキャスト)送信している

- SEDP
  - エンドポイント情報を送受信
    - ACK/NAKで信頼性がある。ユニキャスト
  - Heart beat送信
  - 詳細は次ページ

ROS2プロトコル: SEDP

- SEDP
- 今回の試作はセンサ機器として開発
- 情報受信したいクライアント情報を受信し、管理
  - クライアント・テーブルを作る
  - 新しいクライアントは登録
- 情報送信は、クライアント・テーブルに登録されているクライアントへ順次
  - ラウンドロビン (アプリケーション層の仕事)

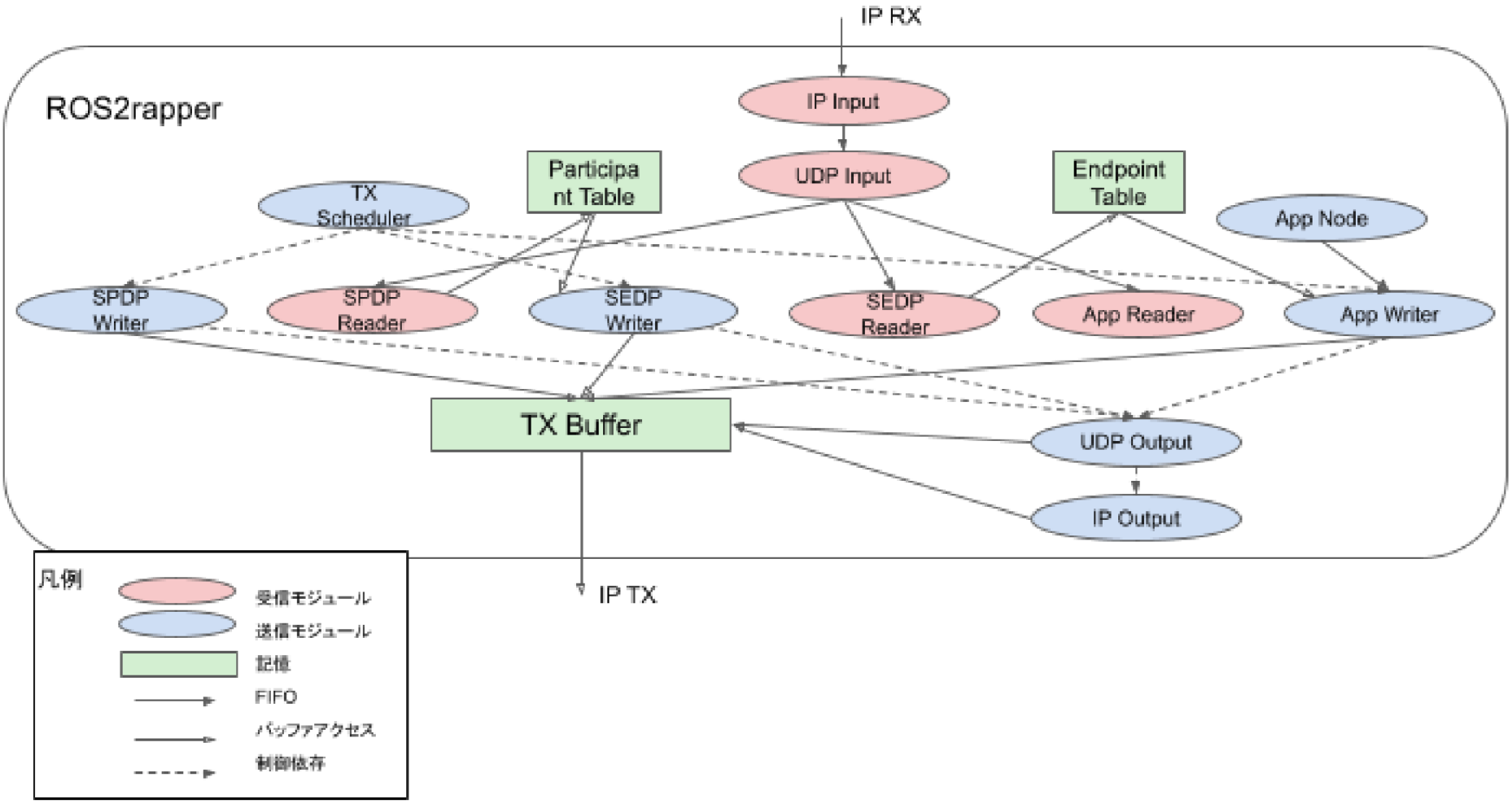


Fig. 2.1　ROS2rapper機能ブロック図

ROS2rapper とユーザ・ロジックのインターフェース

- 主にレジスタ(D-FF)かポート(ワイヤ)で接続
- 1port RAM(など)をユーザ・ロジックで用意
  - D-FFで実現しても良い
  - ROS2は受信用バッファRAMのみ必要
  - UDPを使用する場合は、送受信用のRAMが必要
- Vivado, NEC CWBで合成可能

ROS2rapper 公開中

- GitHubにて公開中
  - https://github.com/AXE-jp/ros2rapper

- ライセンス GPLv3
  - 企業などがGPLを避けたい場合は、当社AXEよりプロプライエタリ版を購入ください

- LSI化にも対応
  - NEC CWBで合成した版も用意
  - オープンソース版も同一ソースなので、CWBで合成可能

付録:
ROS2rapper vs 通常ROS2
比較

2024/NOV/18
AXE, Inc.

ROS2の実行のハードウェア

通常ROS2の実行に必要なマシン

- OS
  - Linux、Windows
- CPU
  - **X86/arm64/arm32 などの高級CPU**が必要
  - **Linuxが動作するMMUなどを装備し、高級かつ高消費電力**
- メモリ
  - 定期的に文字列をパブリッシュするだけの単純なノードでも、**メモリを300MB以上使用**

ROS2rapper

- OS不要
- CPU不要
- メモリ:アプリケーションが必要とする通信バッファのみ
  ※すべての処理は、ハードウェア実装したIPが行う

Architecture	Ubuntu Noble (24.04)	Windows 10 (VS2019)	RHEL 9	macOS	Debian Bookworm (12)	OpenEmbedded / Yocto Project
amd64	Tier 1 [d][a][s]	Tier 1 [a][s]	Tier 2 [d][a][s]	Tier 3 [s]	Tier 3 [s]	Tier 3 [s]
arm64	Tier 1 [d][a][s]				Tier 3 [s]	Tier 3 [s]
arm32	Tier 3 [s]				Tier 3 [s]	Tier 3 [s]

通常ROS2 (Jazzy)がサポートするプラットフォーム

性能比較

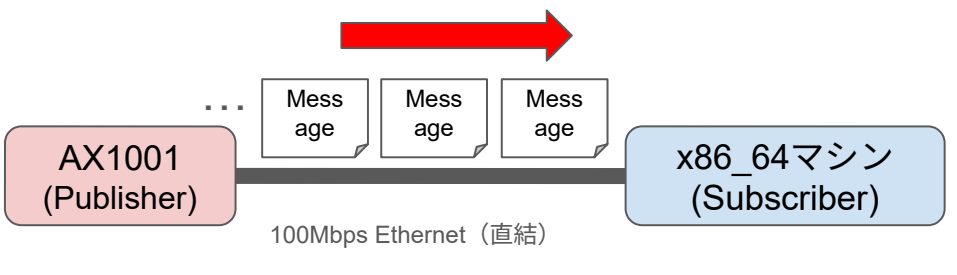
ROS2メッセージの送信速度比較した環境

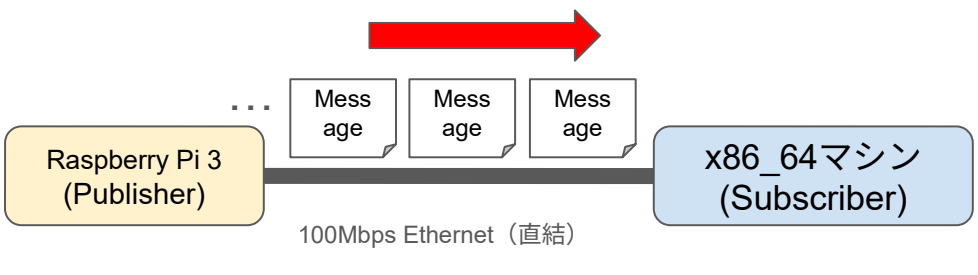
- **AX1001 (ROS2rapperを搭載したSoC) とRaspberry Pi 3 Model Bで、単位時間あたりに送信できるメッセージ数を比較**
- 10000個のROS2メッセージを連続でパブリッシュするのにかかる時間を、受信側(x86_64マシン)のWiresharkのタイムスタンプ情報より測定
  - QoSは、Best effort, Keep last, Depth=0, Volatile
  - x86_64マシン内共有メモリ通信は、送信ノード内で10000個のパブリッシュにかかる時間を測定

AX1001
CPU: 320MHz 32-bit 1-core RISC-V
ROS2rapperのクロック周波数: 80MHz

Raspberry Pi 3 Model B
CPU: 1.2GHz 64-bit ARMv8 Cortex-A53 4-core
Memory: 1GB
OS: Ubuntu Server 22.04 LTS
ROS2 Distro: Humble
DDS: Fast-DDS v2.6.8

x86_64マシン
CPU: x86_64 AMD Ryzen 7 3700X 8-core
Memory: 32GB
OS: Ubuntu 24.04 LTS
ROS2 Distro: Jazzy
DDS: Fast-DDS v2.14.1





ROS2メッセージの送信速度比較 - 測定結果

- AX1001 (ROS2rapper) が1秒間に送信できたメッセージ数は、Raspberry Pi 3の**約8.97倍**

AX1001(ROS2rapper) ---> x86_64マシン（Ethernet UDP通信）	**54223 Messages/sec**
Raspberry Pi 3 ---> x86_64マシン（Ethernet UDP通信）	**6047 Messages/sec**
（参考）x86_64同一マシン上のプロセス間 （共有メモリ通信）	**14556 Messages/sec**

測定結果（10回の平均値）

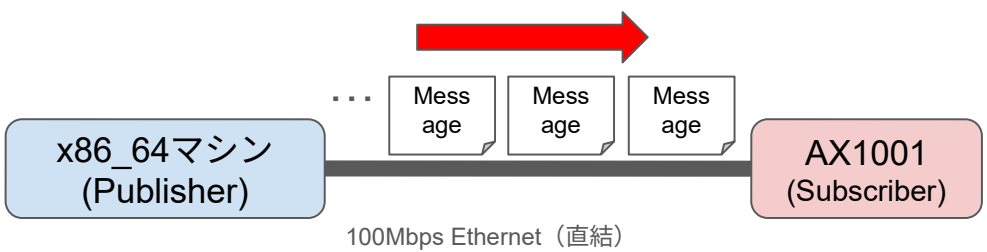
ROS2メッセージの受信速度比較した環境

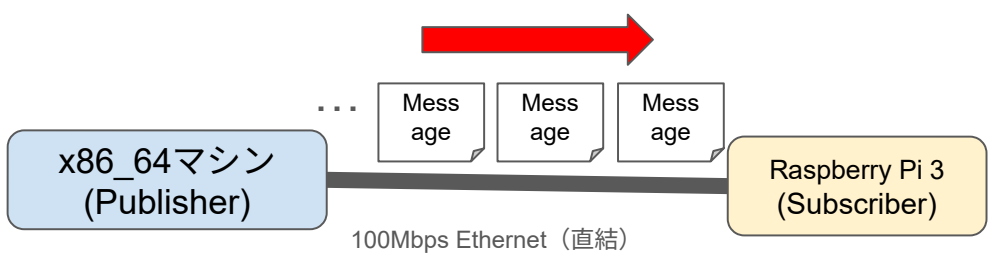
- **AX1001 (ROS2rapperを搭載したSoC) とRaspberry Pi 3 Model Bで、単位時間あたりに受信できるメッセージ数を比較**
- x86_64マシンからROS2メッセージを連続でパブリッシュし、サブスクライバが10000メッセージ受信するのにかかる時間を測定
  - QoSは、Best effort, Keep last, Depth=0, Volatile

AX1001
CPU: 320MHz 32-bit 1-core RISC-V
ROS2rapperのクロック周波数: 80MHz

Raspberry Pi 3 Model B
CPU: 1.2GHz 64-bit ARMv8 Cortex-A53 4-core
Memory: 1GB
OS: Ubuntu Server 22.04 LTS
ROS2 Distro: Humble
DDS: Fast-DDS v2.6.8

x86_64マシン
CPU: x86_64 AMD Ryzen 7 3700X 8-core
Memory: 32GB
OS: Ubuntu 24.04 LTS
ROS2 Distro: Jazzy
DDS: Fast-DDS v2.14.1





ROS2メッセージの受信速度比較 - 測定結果

- AX1001 (ROS2rapper) が1秒間に受信できたメッセージ数は、Raspberry Pi 3の**約2.17倍**

x86_64マシン ---> AX1001(ROS2rapper)（Ethernet UDP通信）	**12201 Messages/sec**
x86_64マシン ---> Raspberry Pi 3（Ethernet UDP通信）	**5635 Messages/sec**
（参考）x86_64同一マシン上のプロセス間 （共有メモリ通信）	**13973 Messages/sec**

測定結果（10回の平均値）

インストール手数の比較

ROS2実行環境のセットアップの手間

通常ROS2実行環境のセットアップ

- 手順が多く、時間もかかる
  - 作業に慣れた者が行っても **3時間** 掛かる
  - Raspberry Piのような非力なマシンだと特に
- インストールサイズは、ROS2だけでも**約2.6GB**と大きい
  - 他の依存パッケージやOSも合わせると、16GBのSDカードでもギリギリ (Raspberry Pi 3の場合)

ROS2rapper
実行環境の構築:不要

アプリケーションIPを合成してハードウェアに流し込むだけ

通常ROS2実行環境のセットアップに要した時間 (Raspberry Pi 3 Model Bの場合)

- SDカードにOSイメージ(Ubuntu Server 22.04 LTS)書き込み (約10分)
- OSのコンフィグ (約10分)
- ROS2リポジトリの追加 (約5分)
  - `sudo apt install software-properties-common && sudo add-apt-repository universe`
  - `sudo apt update && sudo apt install curl -y`
  - `sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg`
  - `echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null`
- ROS2バイナリパッケージのダウンロード (**約30分**)
  - `mkdir -p ~/ros2_humble && cd ~/ros2_humble`
  - `wget https://github.com/ros2/ros2/releases/download/release-humble-20240807/ros2-humble-20240807-linux-jammy-arm64.tar.bz2`
- アーカイブの展開 (**約15分**)
  - `tar xvf ros2-humble-20240807-linux-jammy-arm64.tar.bz2`
- 依存パッケージのインストール (**約2時間**)
  - `sudo apt update && sudo apt install -y python3-rosdep`
  - `sudo rosdep init && rosdep update`
  - `sudo apt upgrade`
  - `rosdep install --from-paths ~/ros2_humble/ros2-linux/share --ignore-src -y --rosdistro humble --skip-keys "cyclonedds fastcdr fastrtps rti-connext-dds-6.0.1 urdfdom_headers"`

合計 約3時間

プログラミング例

ROS2rapper publisher, 初期化パラメータ

```
void main(void)
{
    static const struct ros2_conf conf = {
        .mac_addr = {0x02, 0x00, 0x00, 0x00, 0x00, 0x00},
        .ip_addr = {192, 168, 1, 2},
        .gateway_addr = {192, 168, 1, 1},
        .subnet_mask = {255, 255, 255, 0},
     (中略)
       .node_udpport = 52000,
        .portnum_seed = 0x1ce8,
        .guid_prefix = {0x01, 0x0f, 0x37, 0xad,
                        0xde, 0x09, 0x00, 0x00,
                        0x01, 0x00, 0x00, 0x00},
      (中略)
        .tx_period_spdp_wr     = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_sedp_pub_wr = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_sedp_sub_wr = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_sedp_pub_hb = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_sedp_sub_hb = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_sedp_pub_an = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_sedp_sub_an = ROS2_CLK_FREQ * 3 >> 4,
        .tx_period_app_wr      = ROS2_CLK_FREQ * 3 >> 4,
    };
```

ROS2rapper publisher , 初期化の後、送信メインループ

```
    // initialize
    ros2_set_config(&conf);
    ros2_set_node_name("laxerchip-a");
    ros2_set_pub_topic_name("rt/switches");
    ros2_set_pub_topic_type_name("std_msgs::msg::dds_::String_");

    static char app_data[MAX_APP_DATA_LEN] = "\x0c\x00\x00\x00" "SW0=0 SW1=0";
    app_data[8] = gpio_read(GPIO_SW0) ? '1' : '0';
    app_data[14] = gpio_read(GPIO_SW1) ? '1' : '0';

    ros2_enable_ether();
    ros2_enable_ros2_pub(app_data, sizeof(app_data));

    puts("Initialized.");

    while(1) {
        app_data[8] = gpio_read(GPIO_SW0) ? '1' : '0';
        app_data[14] = gpio_read(GPIO_SW1) ? '1' : '0';
        ros2_set_pub_app_data(app_data, sizeof(app_data));
        task_sleep(task_getid(), CPU_CLK_FREQ >> 10);
    }
}
```

初期化おまじない

データ送信

ROS2rapper subscriber, 初期化の後、受信スレッド

```
void event_handler(void) {
  while (1) {
    extevent_wait(TIMEOUT_INF);
    ros2_request_sub_app_data_grant();
    volatile unsigned char *appdata = ROS2_SUB_APP_DATA_BASE;
    if (ros2_get_sub_app_data_repid() != SP_REPID_CDR_LE) {
      puts("(REPID ERROR)");
    }
    unsigned char len = *ROS2_SUB_APP_DATA_LEN;
    unsigned int slen = appdata[0];
    if (slen > MAX_APP_DATA_LEN - 4) {
      slen = MAX_APP_DATA_LEN - 4;
    }
    printf("length: %d\n", slen);
    puts("data:");
    for (unsigned int i = 0; i < slen; i++) {
      if (appdata[i+4] != '\0')  putc(appdata[i+4], stdout);
    }
    ros2_release_sub_app_data_grant();
    ros2_clear_sub_recv_event();
  }
}
```

ROS受信待ち

データ受信

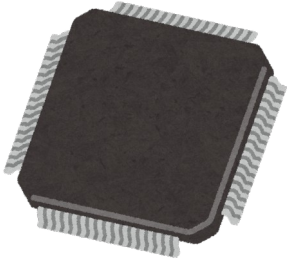
受信バッファ解放

ROS2rapper subscriber, 初期化

```
// initialize
task_start(1, event_handler, 0);
extevent_unmask_event(EVENT_ROS2_SUB_RECV);
extevent_enable();

ros2_set_config(&conf);
ros2_set_node_name("ros2rapper_example");
ros2_set_sub_topic_name("rt/aaa");
ros2_set_sub_topic_type_name("std_msgs::msg::dds_::String_");

ros2_enable_ether();
ros2_enable_ros2_sub();

puts("Initialized.");
```

ROS受信スレッド起動

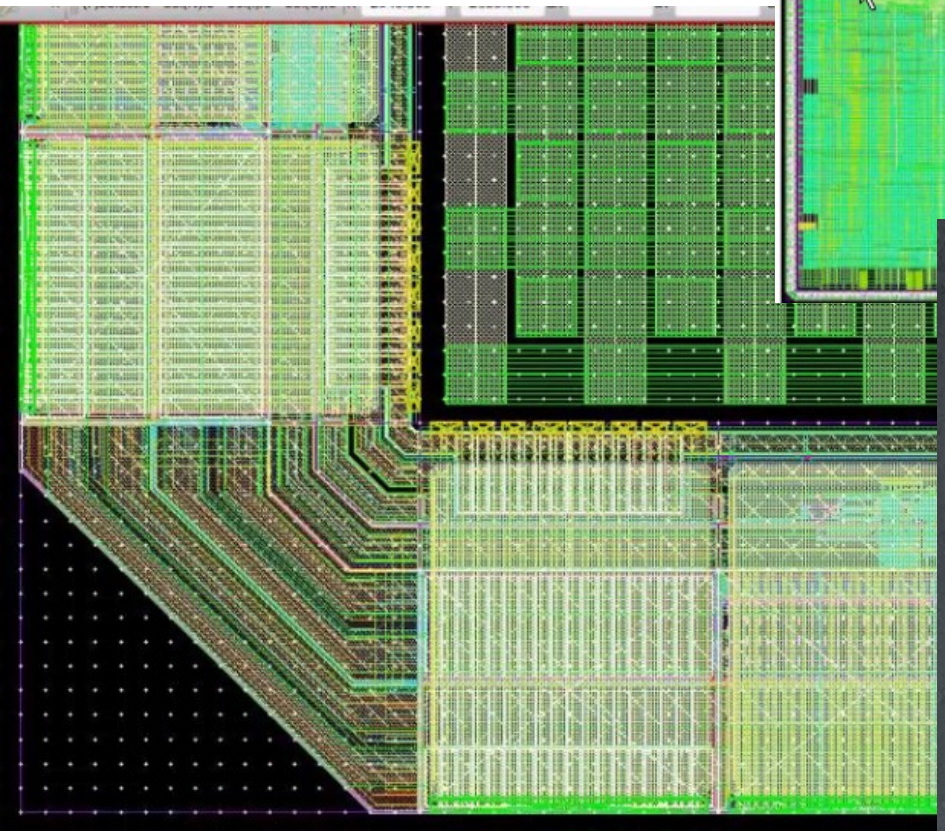
OS無し
ROS2rapper搭載
独自LSI「俺SoC」（AX1001）
の紹介

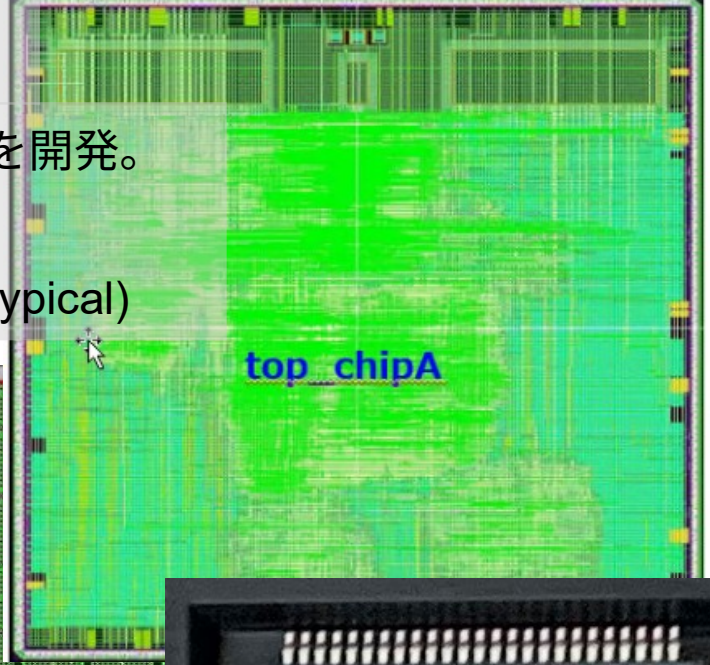


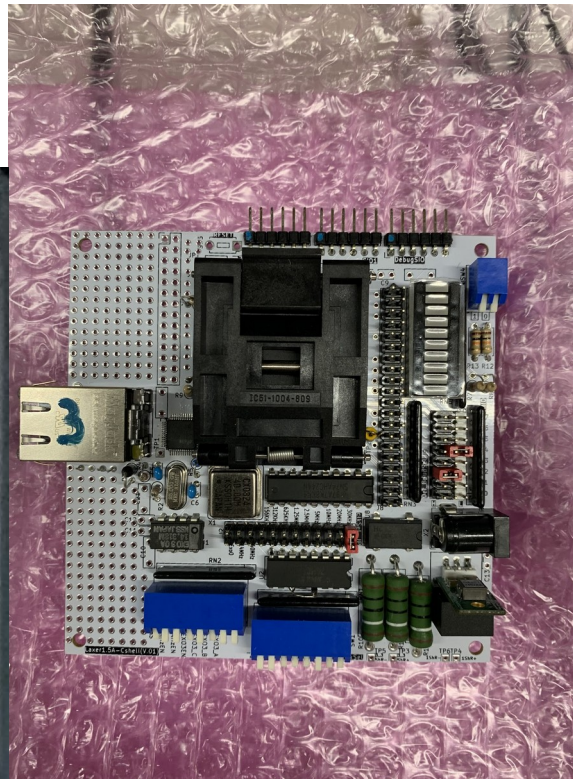
AXEの完全オリジナルLSI AX1001

TSMC 90nmで「俺SoC」LaxerChipを開発。
LQFP 100pin
27/SEP/2024 京都で1stシリコン動作
Clock:480MHz(max),1Hz～320MHz(typical)

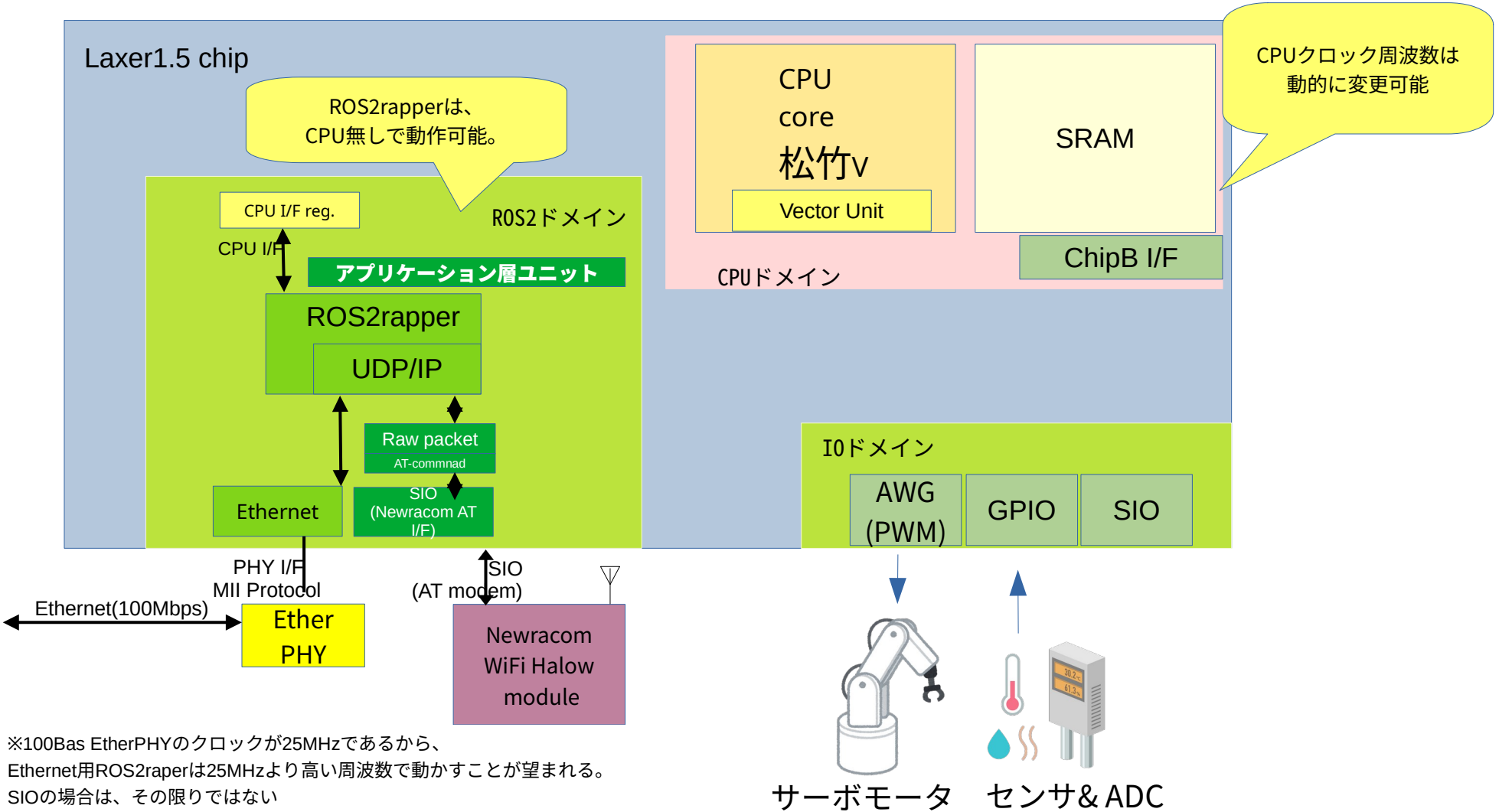
俺SoC







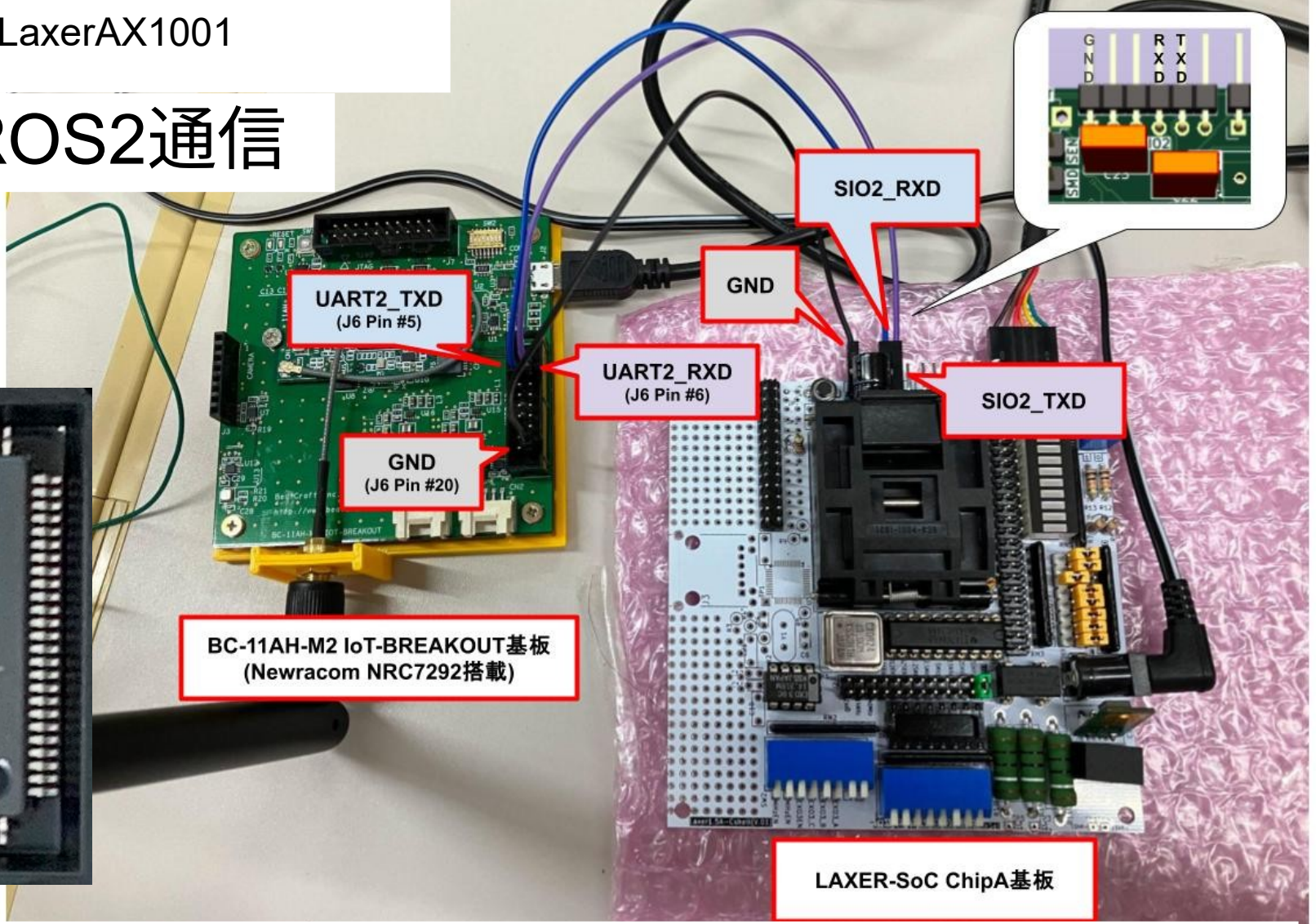
Laxer1.5 ChipA(AX1001)ブロック図



※100Bas EtherPHYのクロックが25MHzであるから、
Ethernet用ROS2raperは25MHzより高い周波数で動かすことが望まれる。
SIOの場合は、その限りではない

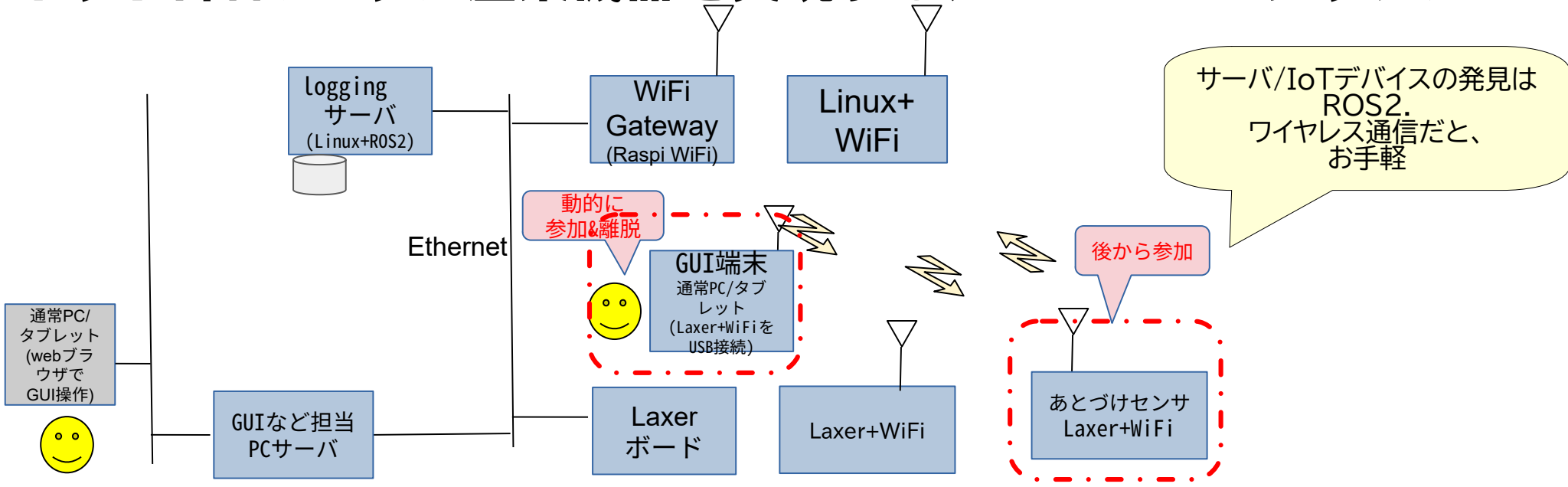
Newracom(802.11ah)+LaxerAX1001

802.11ahでROS2通信



ROS2通信動画 https://drive.google.com/file/d/1WARiMdEghyiwwW-3qw8TR1HwC0N380Ex/view?usp=drive_link

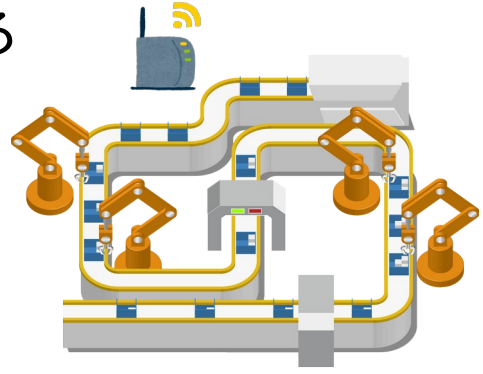
LaxerAX1001は、ROS2通信で、ロボット部品/エッジ産業機器を実現する、マイコン・チップだ



全ノード、ROS2で通信し、動的に参加者(センサなど)を発見できる

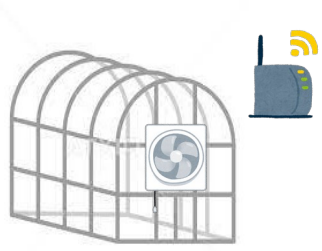
小さなデータであれば、帯域も問題無し
工場、製造業(FA)も、OK

無線センサが、ROS対応だといいね!

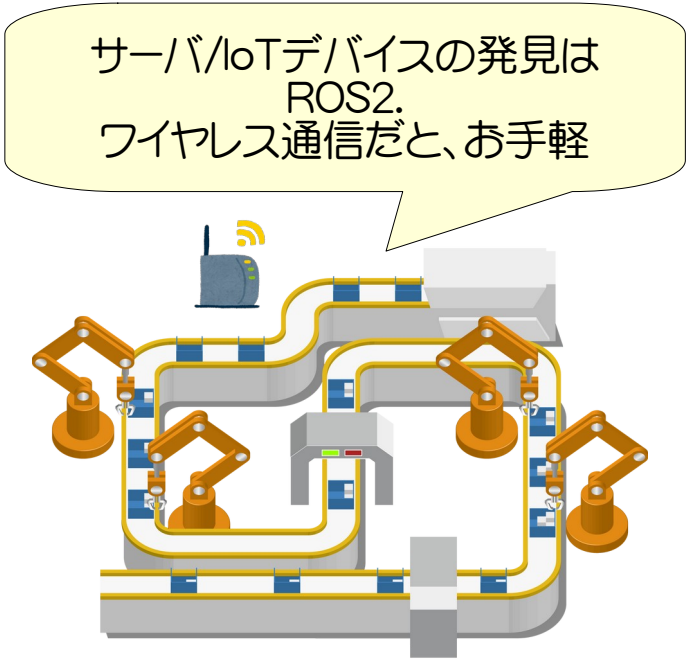


産業のIoTにROSを

- ROS2のadvertisement(広告)とdiscovery(発見)は平易で使いやすい
- ROSのアプリケーションは、(ほぼ)自動で、話し相手を見つけて通信する
- 小さなデータであれば、帯域も問題無し
- 工場、製造業(FA)も、OK
- 無線センサが、ROS対応だといいね!
- バッテリ駆動



モータ (簡素な制御)	土壌センサ/ 温度センサなど…



“俺SoC”,とことんOS無し

- 省電力、省メモリ、堅牢かつ高速
- ロボットの部品モジュールがローコストで

俺SoC

オレ達のCPU「松竹V(しょうちくぶい)」

- 機械学習AI 加速用 ベクトル計算ユニット
  - 8bit float,4SIMD,パイプライン
- 論理推論加速 機構をRISC-Vコアに追加
  - 特許取得済み(第7506718号,2024年6月18日 (東京エレクトロンと共同特許))
  - GnuPrologのコンパイルド・バイナリを加速
- ハードウェア・マルチスレッド機構
  - OSソフトウェア一切なしで、スレッド切り替え
  - ハードウェア・セマフォで排他/同期。LR/SCもある
  - 外部ピンからの入力で、スレッド起床(ハードウェアのみで)
  - 割り込みなし(割り込み相当の処理は、専用スレッドで)

ROS2通信ハードウェア”ROS2rapper”を搭載

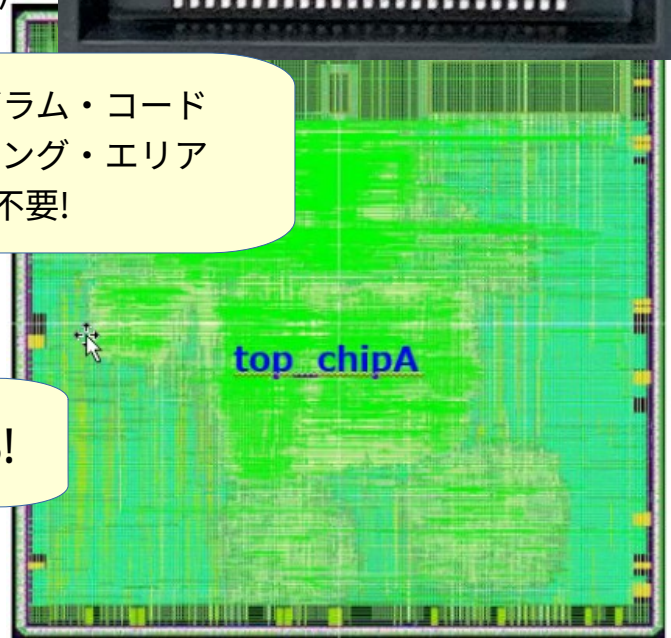
- CPUの助けなしにROS2通信

外部I/O: Ethernet I/F, 任意波形生成器(AWG(PWMにもなる)), GPIO

エッジデバイスでも
大脳的処理を!

OSプログラム・コード
OSワーキング・エリア
不要!

OS 不要のROS!



Laxer AX1001 概要仕様

機能	数	備考
CPU	1	RISC-V RV32Iに、4SIMD 8bit浮動小数点ベクトル機構、ハードウェアによるマルチタスク制御機構、多重分岐命令(特許取得済、Prolog,Lisp,JavaVMの実行を加速)と、それらを操作する命令を付加
RAM	1	プログラム・コードRAM 64KBytes。reset後、bootloaderにより、SIO経由でプログラム・コードを配置可能。CPUコアによるコードRAMの内容変更はできない
	1	データRAM 64KBytes。
SIO(UART)	3	SIO0(UART0)は、bootloadingとデバッグ・コンソールに使用される。UART1,2はユーザが自由に使用できる
GPIO	38	18本はPullUP指定可能。20本はPullDown指定可能。2本は、SIO1と兼用。2本は、SIO2と兼用。18本は外部イベント源として使用可能。(※外部イベントとは、一般CPUにおける割込のようなもの。外部イベントが発生すると、イベントについて待機しているスレッドが起床する)
AWG	1	16bit出力,1ch。出力ピンは、PWMと兼用。PWMと排他的に使用
PWM	8	出力ピンは、AWGと兼用。AWGと排他的に使用
Ether I/F	1	NICはオンチップであり。PHY I/Fを持つ (PHYは外付け)
ROS2通信機能	1	"ROS2rapper"という名称の、新開発のハードウェアによるROS2通信機能。CPUから初期化することで、自動的に通信を行う

松竹V CPU:RECONF講演賞 企業部門@2025年1月研究会 受賞

https://www.ieice.org/iss/reconf/top/?page_id=612

Laxer AX1001 消費電力概要

	テストプログラム名	CPUクロック周波数・消費電力(mW)					
-	-	320MHz			1.79MHz		
-	-	実測 1.0v(mA)	実測 3.3v(mA)	1.0V+3.3v計 (mW)	実測 1.0v(mA)	実測 3.3v(mA)	1.0V+3.3v計 (mW)
	メモリテスト	177.0	5.0	193.5	10.0	0.0	10.0

参考	※実際には、DC-DCコンバータなどで損失が出るので、仮の理想値	電池容量 (mW換算)	320MHz(時間)	1.79MHz(時間)
	単1形アルカリ電池容量 約10,000 mAh/1本, 3本4.5v使用	135,000	697.67	13,500.00
	単3形アルカリ電池 容量約1,000～2,900mAh/1本, 3本4.5v使用	39,150	202.33	3,915.00
	コイン電池CR2450（3V) 620mAh, 2個(6V)使用 （※最大電流30mA/1個)	7,440	(38.45 ※電流が足りない)	744.00

表 5　LAXER-SoC の消費電力

動作状態	消費電力 (mW)		
	1V コア電源	3.3V IO 電源	合計
全タスク停止時	117	10	127
1 タスク動作時	138	10	148
4 タスク動作時	145	9	154
浮動小数点数ベクトル演算時	136	10	146
ROS2 Publisher 動作時	119	10	129

松竹 V のモジュールごとのゲート数およびその割合

階層名	ゲート数	割合 (%)
松竹 V	428,062	100.00
整数レジスタ	46,511	10.87
タスク管理ユニット	5,499	1.28
ベクトルユニット	347,699	81.23
ベクトルレジスタ	325,345	76.00
8-bit 浮動小数点数演算器	15,478	3.62
整数演算器	1,667	0.39
ALU	1,896	0.44
BTB	508	0.12
PHT	1,542	0.36

オレ達のCPU「松竹V(しょうちくぶい)」の排他制御

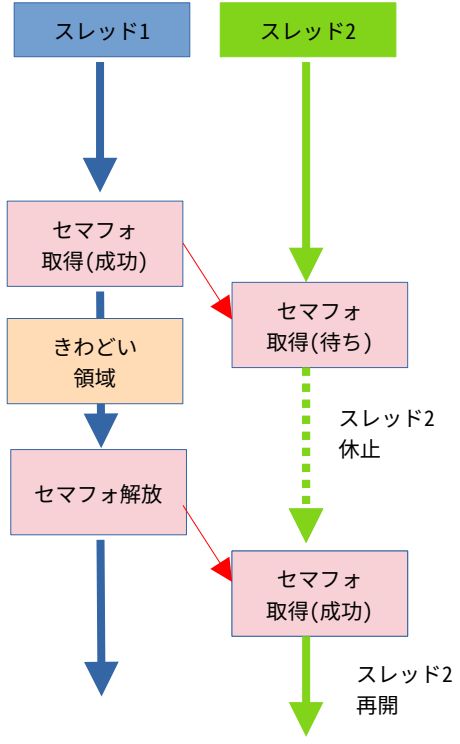
“俺SoC”,とことんOS無し

- **省電力、省メモリ、堅牢かつ高速**
- **ロボットの部品モジュールがローコストで簡単に作れる**

マルチタスクだが、
OSプログラム・コード
OSワーキング・エリア
不要!

ハードウェア・マルチスレッド機構

- OSソフトウェア一切なしで、スレッド切り替え
- ハードウェア・セマフォで排他/同期(重要)
- LR/SC(RISC-Vの排他制御プリミティブ)もある
- 外部ピンからの入力で、スレッド起床(ハードウェアのみで)
  - 割り込みなし(割り込み相当の処理は、専用スレッドで)
- ラウンドロビン・スケジューリング

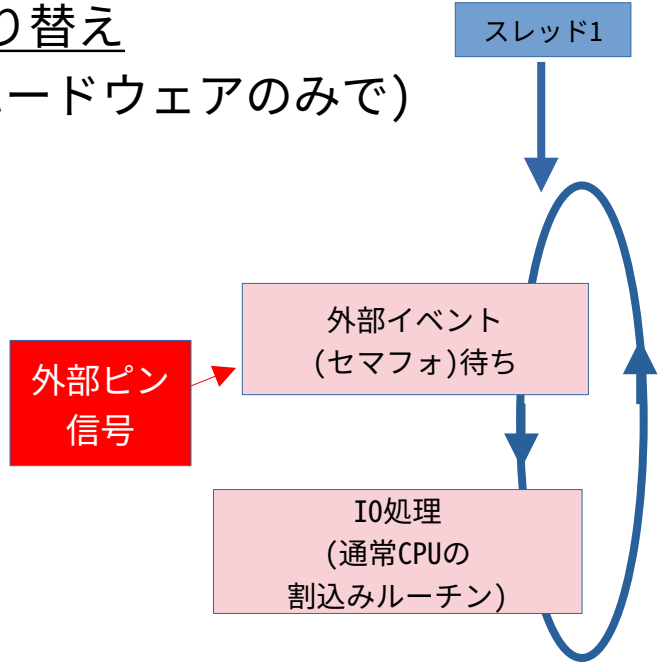


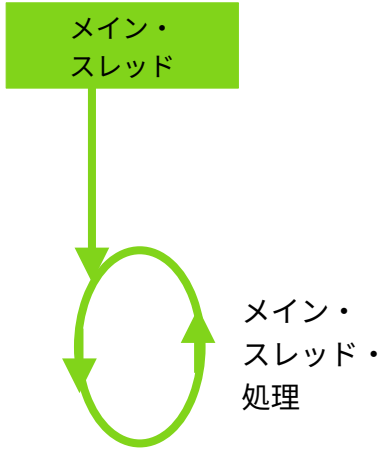
オレ達のCPU「松竹V(しょうちくぶい)」の外部イベント（割込み相当）

マルチタスクだが、
OSプログラム・コード
OSワーキング・エリア
不要!

ハードウェア・マルチスレッド機構

- OSソフトウェア一切なしで、スレッド切り替え
- 外部ピンからの入力で、スレッド起床(ハードウェアのみで)
  - 割り込みなし
- 割り込み相当の処理は、専用スレッドで。
- 割込み起動より、高速
  - レジスタ退避などしないから





スレッド操作、セマフォ操作命令

6.5.7. カスタム命令

ニーモニック	命令フォーマット	opcode	funct3	funct7	imm
task_start	R-type	01010 (custom-1)	000	0000000	-
task_terminate	R-type	01010 (custom-1)	000	0000001	-
task_getid	R-type	01010 (custom-1)	000	0000010	-
task_read_xreg	R-type	01010 (custom-1)	000	0000011	-
task_write_xreg	R-type	01010 (custom-1)	000	0000100	-
task_sleep	R-type	01010 (custom-1)	000	0000101	-
task_yield	R-type	01010 (custom-1)	000	0000110	-
sem_init	R-type	01010 (custom-1)	000	0000111	-
sem_signal	R-type	01010 (custom-1)	000	0001000	-
sem_wait	R-type	01010 (custom-1)	000	0001001	-
task_setts	R-type	01010 (custom-1)	000	0001010	-

オレ達のCPU「松竹V(しょうちくぶい)」

エッジデバイスでも
AI処理を!

機械学習AI 加速用 ベクトル計算ユニット

・8bit float,4SIMD,ベクトル・パイプライン
・動的クロック切り替え機構で500KHz～320MHzで、動作
※最高 480MHz (動的クロック切り替え非対応)
・Vector ExtensionのImplementation-defined Constant Parameters(実装固有パラメータ)
ELEN:32 ,VLEN: 1024

・ベクトルレジスタ
1024ビットのベクトルレジスタ×32個
ベクトルレジスタはすべてのタスクで共有(実体は1つ)
機械学習AIの推論に最適な8bit浮動小数点演算をベクトル処理

- エッジ機器内での、AI処理
↓
- データ通信量の削減

・分散処理による、全体の負荷の分散

32ビットのスカラ浮動小数点数レジスタ(Fレジスタ): 32個も備える

8bit浮動小数点 ベクトル演算命令

- AX1001の8bit浮動小数点数のフォーマット
  - 仮数部:3bit, 符号:1bit 指数部:4bit, 指数バイアス=7

6.7.8. サポートするCSR

Vector Extensionで規定される、以下のCSRをサポートしている。

CSRアドレス	アクセス	名前	概要
0xC20	R	vl	Vector length
0xC21	R	vtype	Vector data type register
0xC22	R	vlenb	VLEN/8 (vector register length in bytes)

6.7.6. サポートする命令

- V拡張
  - vsetvli
  - vsetivli
  - vsetvl
  - vle.v
  - vse.v
  - vlse.v
  - vsse.v
  - vlm.v
  - vsm.v
  - vadd.{vv, vx, vi}
  - vsub.{vv, vx}
  - vand.{vv, vx, vi}
  - vor.{vv, vx, vi}
  - vxor.{vv, vx, vi}
  - vsll.{vv, vx, vi}
  - vsrl.{vv, vx, vi}
  - vsra.{vv, vx, vi}
  - vfadd.{vv, vf}
  - vfsub.{vv, vf}
  - vfmul.{vv, vf}
- F拡張
  - flw

オレ達のCPU「松竹V(しょうちくぶい)」

- **省電力、省メモリ、堅牢かつ高速**
- **ロボットの部品モジュールがローコストで簡単に作れる**

論理推論 AI加速 機構をRISC-Vコアに追加

- 特許取得済み
- 第7506718号,2024年6月18日（東京エレクトロンと共同特許)

- GnuPrologのコンパイルド・バイナリを加速
  - 詳細は、付録参照のこと

エッジデバイスでも
大脳的処理を!

その他のカスタム命令

6.4.2. カスタム命令

ニーモニック	命令フォーマット	opcode	funct3	funct7	imm
mov_reg_regindirect	I-type	00010 (custom-0)	000	-	Don't care
mov_regindirect_reg	I-type	00010 (custom-0)	001	-	Don't care
branch_reg_indirect	I-type	00010 (custom-0)	010	-	Don't care
jump_reg_indirect	R-type	00010 (custom-0)	011	0000000	Don't care

6.4.2.3. branch_reg_indirect

プログラムカウンタ相対で分岐を行う。プログラムカウンタに加算する値をソースオペランドをレジスタ間接で指定する。後続命令のアドレス（pc+4）をレジスタrdに格納する。

本命令は、次の操作を行う。

```
x[rd] <- pc + 4
pc <- pc + x[x[rs1]]
```

6.4.2.4. jump_reg_indirect

分岐を行う。分岐先のベースアドレスをレジスタ間接で指定する。後続命令のアドレス（pc+4）をレジスタrdに格納する。

本命令は、次の操作を行う。

```
rd <- pc + 4
pc <- x[x[rs1]] + x[rs2]
```

以上