ロボット&IoTエッジ向け
「俺SoC」
AX1001について

2025/AUG/29

たけおか@AXE　@takeoka

（2009 SWEST11「仮想化 技術について」以来2回目）

(株)アックス(創業33年)の基本ソフトの採用実績

シャープ
携帯電話

オリンパス デジカメ

ザウルス

シャープ ホームサーバ

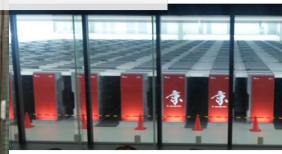
ロボット

（産総研）

航空自衛隊

自動運転Autoware

国スパコン富岳 OS研究



富岳にはMcKernelが採用されている

XcalableMPの仕様策定に参加

同言語コンパイラ開発

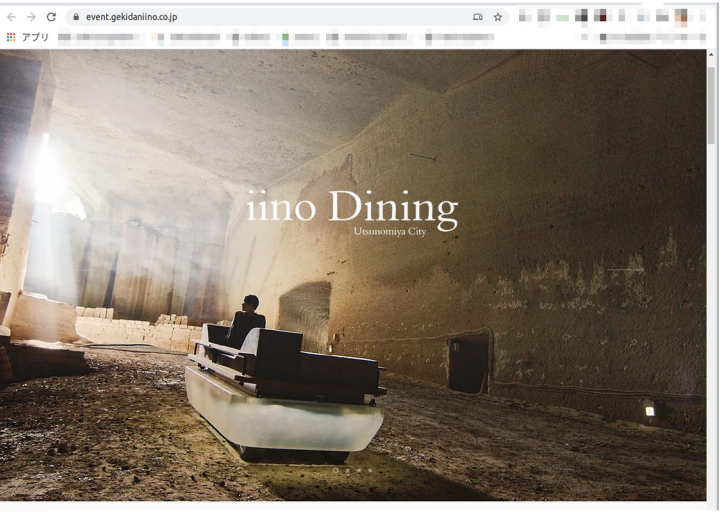
資本金+資本準備金=5億3千万円
Nextyエレ（トヨタグループ）からも資本が入っていた
村井純先生（インターネット殿堂入り）も株主だ

・日本の独自CPUの基本ソフトウェアをサポート
富士通 FR/V,ルネサス(旧日立製作所)SH-Mobile,SH2A, 東芝MeP,
ルネサス(旧NEC)V850, セイコーエプソンC33,C17、シャープLH795xx

Autoware(ROS使用)応用 iinoプロジェクト

- iino(ゲキダン イイノ)
  - 関西電力の新規事業プロジェクト
  - 時速 5km/h でゆっくり走行

https://gekidaniino.co.jp/
https://iinomob.jp/



宇都宮 大谷資料館(採石場跡地)

https://takanawa-iino.com/

2023年11月 御堂筋で公道走行

https://gekidaniino.co.jp/wp/wp-content/uploads/2023/11/%E3%83%88%E3%83%83%E3%83%97%EF%BC%92%E3%83%9F%E3%83%8B.pngより引用

2025年3月27日～

- 街なかを移動
  - チョイ乗り

ハイパ岡田メソッドAI

- サッカー・コーチングAI

- エキスパート・システム

- 岡田 元監督たちと会議
  - サッカー&岡田メソッドについて教えてもらう
  - 根本的な疑問をぶつけ、→ 本の改訂につながる
  - コーチ陣は、岡田メソッドに忠実に基づいて思考していてブレが無い

岡田氏談:

「システムエンジニアは凄い、“マーク”とは？と聞かれて、それを具体的な数字で表す話まで落としていったり、目線や体の向きまでもデータとして取り込んでメソッドと関連付けてしまう」

「監督は、その場で状況を分析して判断している。試合のハーフタイムでデータアナリストの分析結果を聞いても役に立たない。AIで（リアルタイムに）データ解析して判定出来れば、監督の目が届かない、気が付かない部分まで網羅できるようになる」

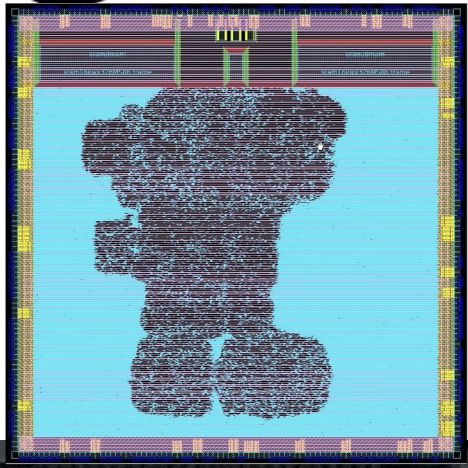
「AXEと言う会社の論理推論型AIと言う凄い技術で、岡田メソッドそのものをAI化する取り組みをやっている」

AXEの完全オリジナルLSI ”Laxer AX1001” 概要仕様

TSMC 90nmで製造,920万トランジスタ
LQFP 100pin
27/SEP/2024 京都で1stシリコン動作

俺SoC

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。CPU Clock=1Hz～320MHz, (特殊モード480MHz)
RAM	1	プログラム・コードRAM 64KBytes+データRAM 64KBytes
SIO(UART)	3	SIO0(UART0)は、起動時に使用される。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。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から初期化後、自動的に通信を行う

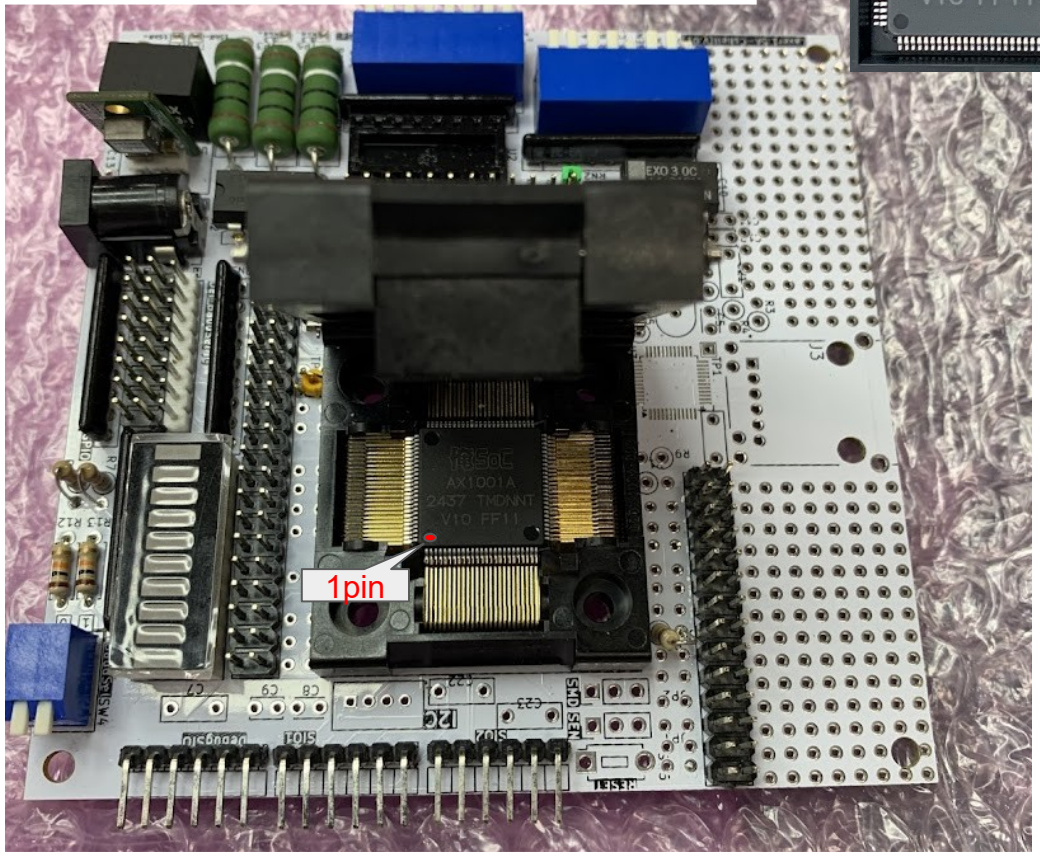


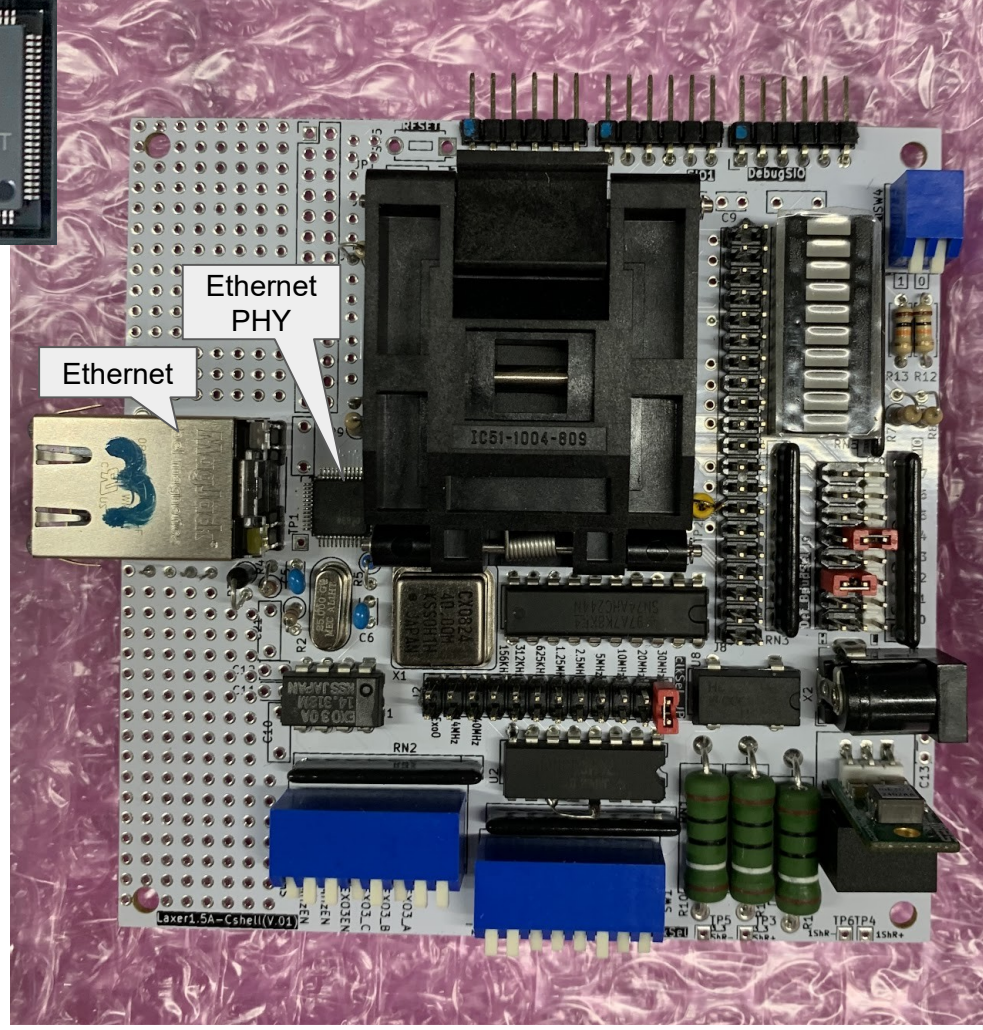
ChipAのレイアウト

松竹V :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

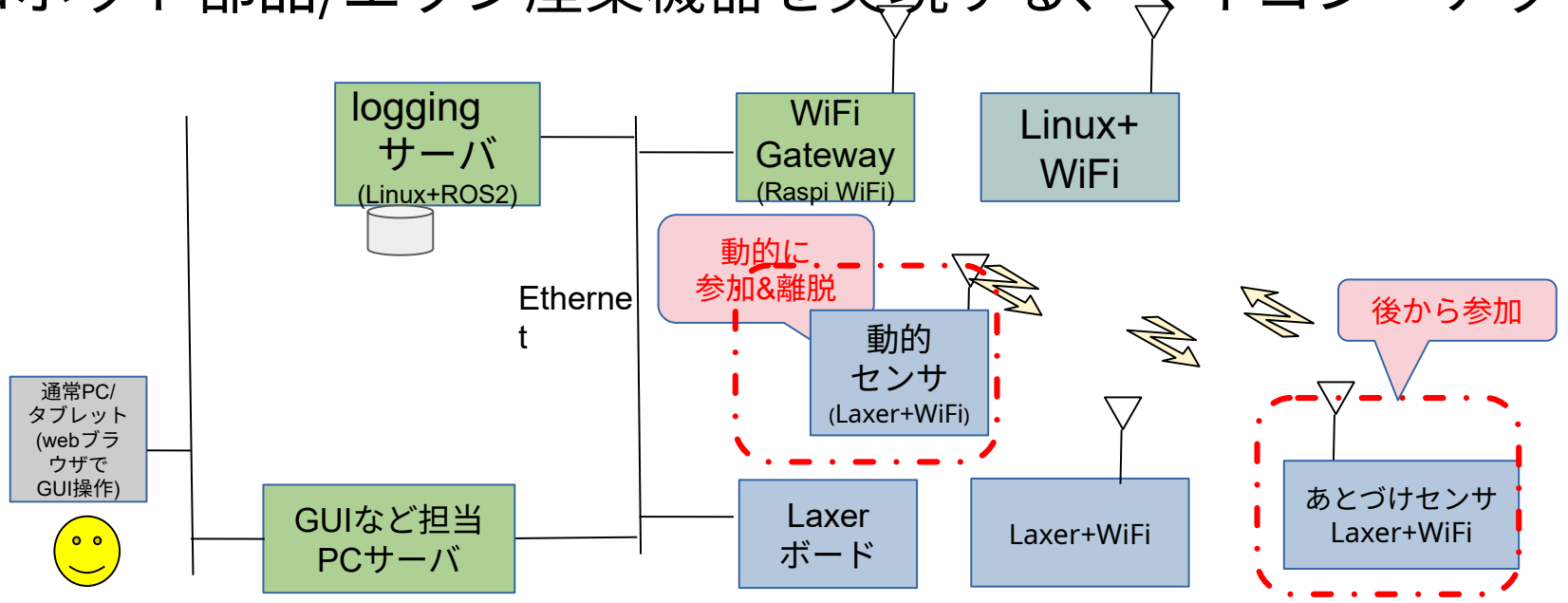
テスト基板
Ethernetで、ハードウェアROS2が動作





LEDチカチカ https://drive.google.com/file/d/10z89qI9ZQaNe8txQFIMjL5HRLoI7bTh4/view?usp=drive_link
ROS2Subscribe https://drive.google.com/file/d/1M2Jt1u0B1dGReELrE8qoQbPvnz0AZl3c/view?usp=drive_link
ROS2Publish https://drive.google.com/file/d/1CCb1PVTXruBFXhebgIjcXFeVvfWzMIoC/view?usp=drive_link

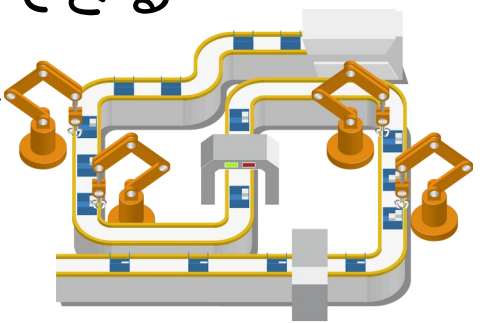
LaxerAX1001は、ROS2通信で、
ロボット部品/エッジ産業機器を実現する、マイコン・チップだ



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

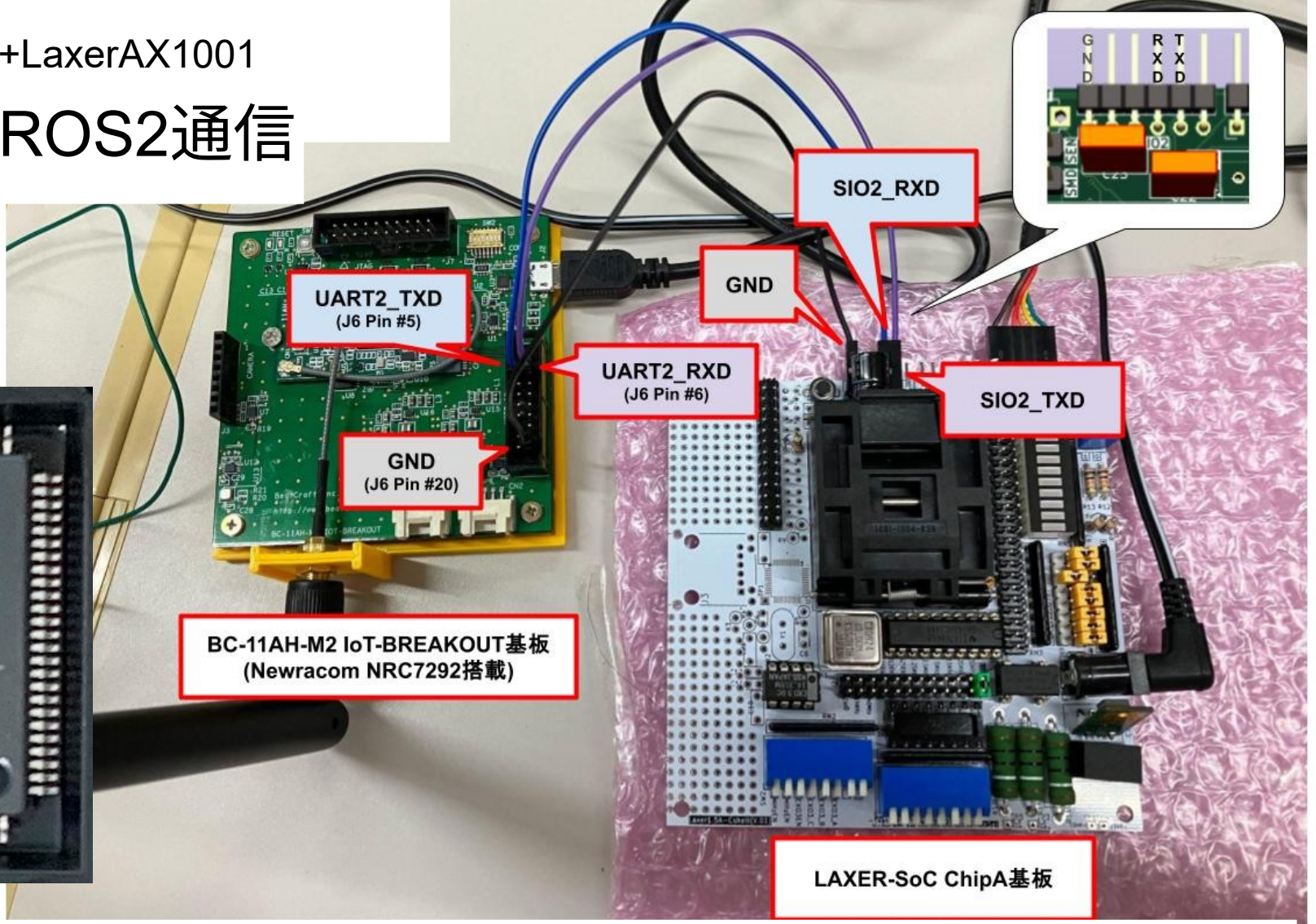
小さなデータであれば、帯域も問題無し
工場、製造業(FA)も、OK

無線センサが、ROS対応だといいね!



WiFi HaLow(802.11ah)+LaxerAX1001

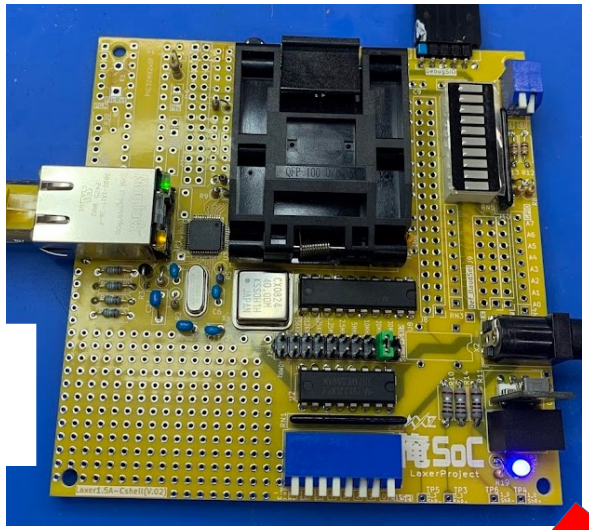
- 802.11ahでROS2通信



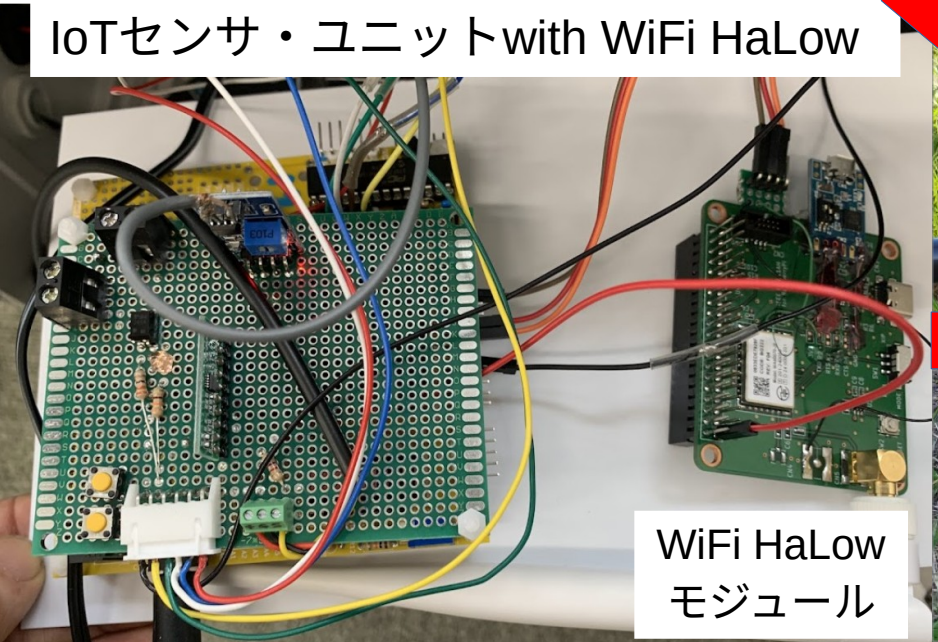
ROS2通信動画 https://drive.google.com/file/d/1WARiMdEghyiwwW-3qw8TR1HwC0N380Ex/view?usp=drive_link

農業エッジIoT PoC

AX1001
評価基板(V2)



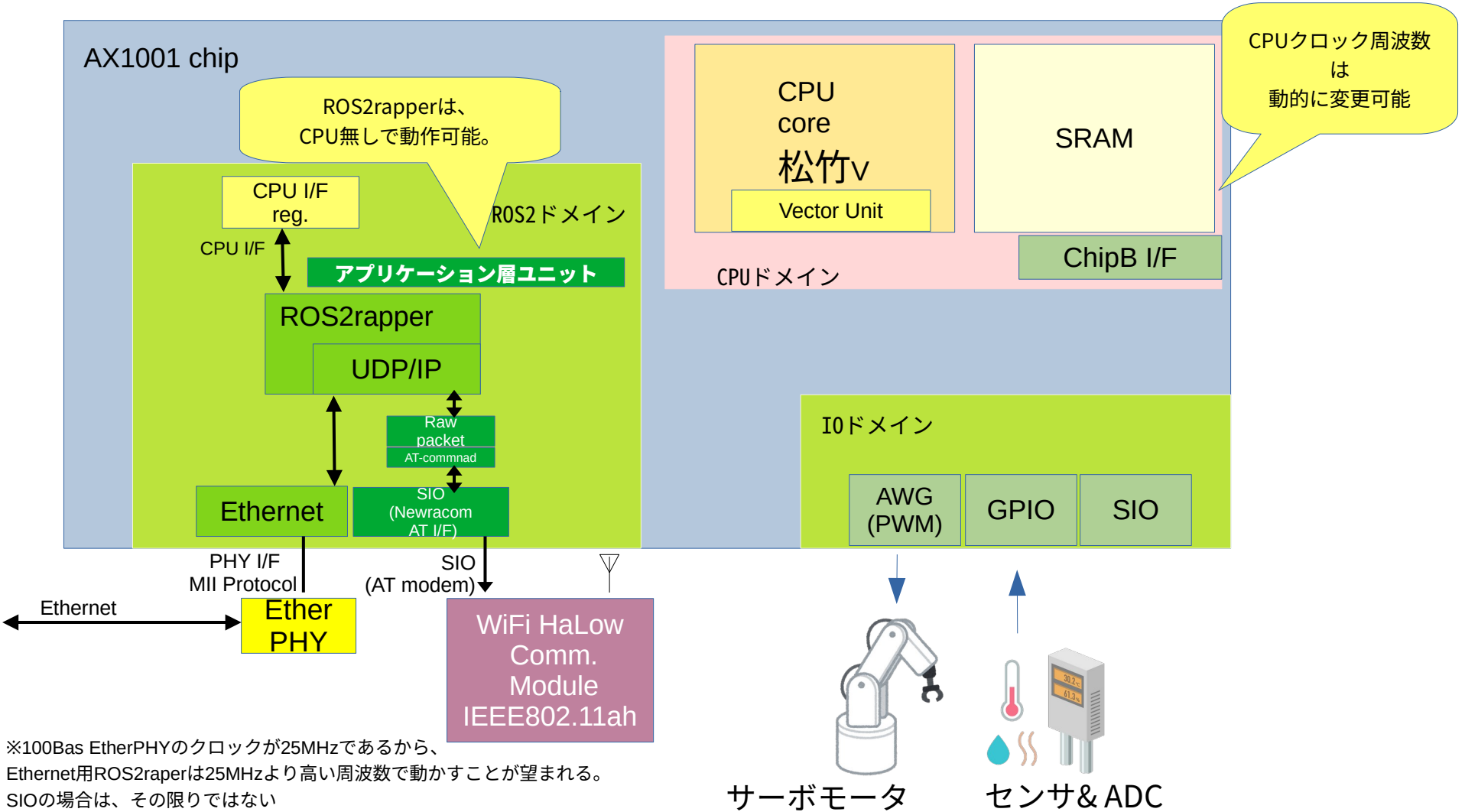
IoTセンサ・ユニットwith WiFi HaLow



WiFi HaLow
モジュール

太陽電池駆動のIoTセンサ群

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

“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

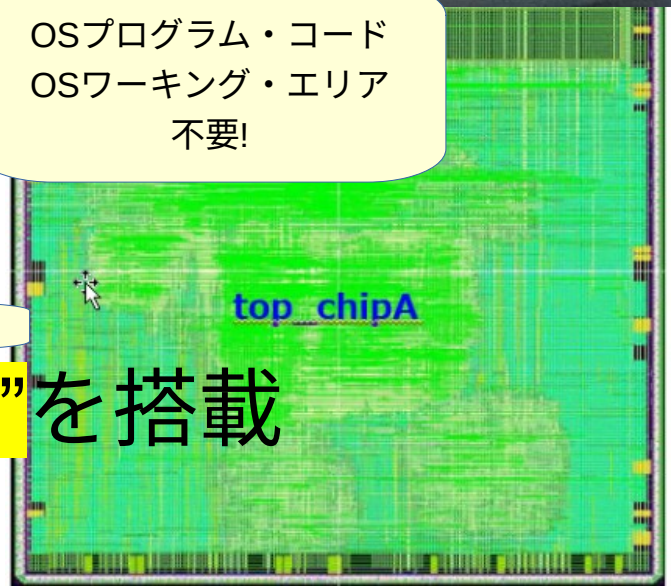
俺SoC

オレ達のCPU「松竹V(しょうちくぶい)」

- 東京科学大 吉瀬研 “RVCoreP” をベースに独自に拡張
- 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
- 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)

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!

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

Laxer AX1001 消費電力概要

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 (Ethernet)	119	10	129

@320MHz

		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

松竹V CPU 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

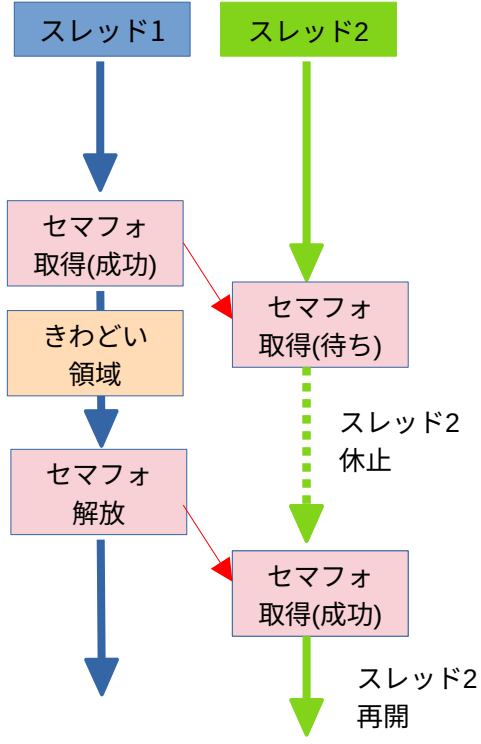
“俺SoC”,とことんOS無し

- **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)
  - セマフォごとに、待ちキューを持つ
  - 待ちはタイムアウト付き
- 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

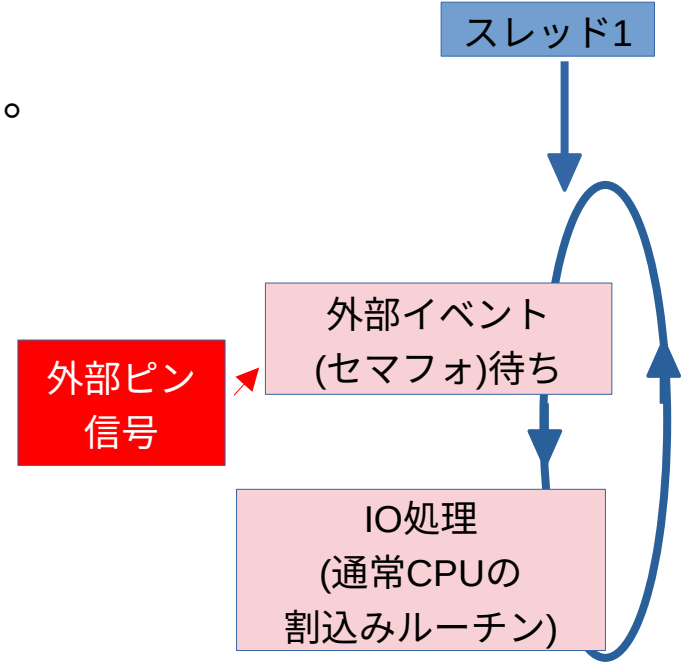


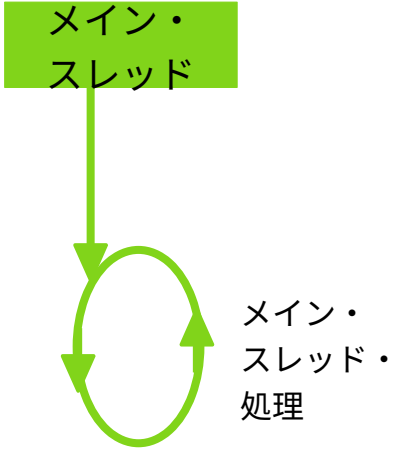
オレ達のCPU「松竹V(しょうちくぶい)」の外部イベント(割込み相当)

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)」

機械学習AI 加速用 ベクトル計算ユニット

エッジデバイスでも
AI処理を!

・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!

Ruby,Haskell, Lispなど動的にオブジェクトの型を判定する 高級な言語は同様に高速化可能

MIT Lisp MachineやPSI(ICOTのPrologマシン,1986年)を超えたいぞ

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]
```

複数タスク実行時の分岐予測ヒット率の評価

- タスク数1→2で、予測ヒット率が12-16ポイント低下
- 異アドレスのプログラムは、同アドレスに比べ、3-4ポイント低下
- ⇨タスク間で分岐予測テーブルを共有するので、アドレスが異なるプログラムだとヒット率が低下したと考えられる

プログラム	分岐回数	予測ヒット率 (%)
2000要素のクイックソート		
1タスク	54,723	63.3
2タスク(同アドレスのプログラム)	109,437	51.3
2タスク(異アドレスのプログラム)	109,439	48.0
8-queen		
1タスク	51,533	70.4
2タスク(同アドレスのプログラム)	103,060	58.3
2タスク(異アドレスのプログラム)	103,052	54.4

※ 同アドレス... 2タスクが同じ関数を呼び出す

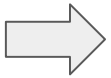
※ 異アドレス... 内容が同じでアドレスが異なる関数を呼び出す

新命令により
論理推論AI(Prolog)を
高速化

GNU-PrologのWAMコード

- GNU-Prolog の WAM コードでは、switch_on_term で 1つめの arity の型(変数/アトム/整数/リスト/構造体)によって分岐する
- 分岐先で各arityの値をチェックする

```
foo(1, a, X) :- bar(1, X).
foo(a, [c,d], X) :- bar(2, X).
foo([a,b], c(d), X) :- bar(3, X).
foo(a(b), 1, X) :- bar(4, X).
```



```
predicate(foo/3,1,static,private,monofile,global,[
    switch_on_term(1,4,2,6,8),

label(1),
    try_me_else(3),

label(2),
    get_integer(1,0),
    get_atom(a,1),
    put_value(x(2),1),
    put_integer(1,0),
    execute(bar/2),

label(3),
    retry_me_else(5),

label(4),
    get_atom(a,0),
    get_list(1),
    unify_atom(c),
    unify_list,
    unify_atom(d),
    unify_nil,
    put_value(x(2),1),
    put_integer(2,0),
    execute(bar/2),

label(5),
    retry_me_else(7),

label(6),
    get_list(0),
    unify_atom(a),
    unify_list,
    unify_atom(b),
    unify_nil,
(以下略)
```

Pl_Switch_On_Term()の内容

ランタイム関数Pl_Switch_On_Term() では、1つめのarityであるA(0)のタグ(下位3ビット)によって、次に実行する分岐先を返すようになっている

```
CodePtr FC
Pl_Switch_On_Term(CodePtr c_var,
   CodePtr c_atm, CodePtr c_int,
   CodePtr c_lst, CodePtr c_stc)
{
  WamWord word, tag_mask;
  CodePtr codep;

  DEREF(A(0), word, tag_mask);
  A(0) = word;

  if (tag_mask == TAG_INT_MASK)
    codep = c_int;
  else if (tag_mask == TAG_ATM_MASK)
    codep = c_atm;
  else if (tag_mask == TAG_LST_MASK)
    codep = c_lst;
  else if (tag_mask == TAG_STC_MASK)
    codep = c_stc;
  else  /* REF or FDV */
    codep = c_var;

  return (codep) ? codep : ALTB(B);
}
```

RISC-V 64bit Gnu Prolog コンパイルド・バイナリとその高速化

分岐先 候補のアドレスを
各 汎用レジスタにロード

gplcでコンパイルした native codeをobjdump -d の一部

```
00000000000096b8 <X0_ap__a3>:
    96b8: 00000517      auipc a0,0x0
    96bc: 02050513      addi a0,a0,32 # 96d8 <X0_ap__a3+0x20>
    96c0: 00000597      auipc a1,0x0
    96c4: 02458593      addi a1,a1,36 # 96e4 <X0_ap__a3+0x2c>
    96c8: 00000617      auipc a2,0x0
    96cc: 05460613      addi a2,a2,84 # 971c <X0_ap__a3+0x64>
    96d0: 014900ef      jal ra,996e4 <Pl_Switch_On_Term_Var_Atm_Lst>
    96d4: 00050067      jr a0
    96d8: 00000517      auipc a0,0x0
    96dc: 04050513      addi a0,a0,64 # 9718 <X0_ap__a3+0x60>
    976c: 00ab3823      sd a0,16(s6)
    9770: f49ff06f      j 96b8 <X0_a
    9774: 00000013      nop
```

レジスタ間接によるレジスタ指定ができる新命令を追加
特にジャンプ命令
ロード命令、ストア命令

例えば、WAMコード switch_on_term をアセンブリ・コードにする場合に、型を示すタグ(3bit)によって、レジスタ間接によって指定されたレジスタの値(アドレス)にジャンプすると、高速になる
すなわち

jal ra,Pl_Switch_On_Term@PLT ;サブルーチン・コールなので、とても遅い

jr a0

を、下記1命令で置き換えて、高速化

branch_reg_indirect IREG ; pc ← pc + xreg[IREG]

この2行の部分を、
あるレジスタ中の値で指定された、
別なレジスタ中の値を番地として、
jumpするようなコードに変更する
(コンパイラが新命令を使用するように変更)
新命令「branch_reg_indirect %rax」
で置き換えて、高速化

Ruby,Haskell, Lispなど動的にオブジェクトの型を判定する 高級な言語は同様に高速化可能

参考:高速化対象: GNU-Prologの80x86(64bit)版オブジェクト(アセンブリ・コード)

この2行の部分を、
あるレジスタ中の値で指定された、
別なレジスタ中の値を番地として、
jumpするようなコードに変更する
(コンパイラが新命令を使用するように変更)
新命令「 branch_reg_indirect %rax 」
で置き換えて、高速化

```
X0_foo__a3:
        movq    Lpred1_1@GOTPCREL(%rip),%rdi
        movq    Lpred1_4@GOTPCREL(%rip),%rsi
        movq    Lpred1_2@GOTPCREL(%rip),%rdx
        movq    Lpred1_6@GOTPCREL(%rip),%rcx
        movq    Lpred1_8@GOTPCREL(%rip),%r8
        call    PI_Switch_On_Term@PLT
        jmp     *%rax

Lpred1_1:
        movq    Lpred1_3@GOTPCREL(%rip),%rdi
        call    PI_Create_Choice_Point3@PLT

Lpred1_2:
        movq    $15,%rdi
        movq    0(%r12),%rsi
        call    PI_Get_Integer_Tagged@PLT
        test    %rax,%rax
        je      fail
        (略)
        jmp     X0_bar__a2@PLT

Lpred1_3:
        movq    Lpred1_5@GOTPCREL(%rip),%rdi
        call    PI_Update_Choice_Point3@PLT

Lpred1_4:
        movq    ta@GOTPCREL(%rip),%rdi
        movq    0(%rdi),%rdi
        movq    0(%r12),%rsi
        call    PI_Get_Atom_Tagged@PLT
        test    %rax,%rax
        je      fail
        (略)
        jmp     X0_bar__a2@PLT

Lpred1_5:
        movq    Lpred1_7@GOTPCREL(%rip),%rdi
        call    PI_Update_Choice_Point3@PLT

Lpred1_6:
        movq    0(%r12),%rdi
        call    PI_Get_List@PLT
        test    %rax,%rax
        je      fail
(以下略)
```

- GNU-Prolog のアセンブリコードでは、WAM(Prolog中間 仮想マシン)のswitch_on_termに対応するランタイム関数PI_Switch_On_Term() を呼んでいる
- その返り値でジャンプしている

ここで、

- レジスタ間接によるレジスタ指定ができる新命令を追加
  - 特にジャンプ命令
  - ロード命令、ストア命令
- 例えば、WAMコード switch_on_term をアセンブリ・コードにする場合に、型を示すタグ(3bit)によって、レジスタ間接によって指定されたレジスタの値(アドレス)にジャンプすると高速になる
- すなわち

```
call    PI_Switch_On_Term@PLT ;サブルーチン・コールなので、とても遅い
jmp     *%rax
```

- を、下記1命令で置き換えて、高速化

branch_reg_indirect IREG ; pc ← pc + xreg[IREG]

レジスタ間接レジスタ指定 命令

【特許第7421850号】
登録日【2024年1月17日】

- branch_reg_indirect IREG

  pc ← pc + xreg[IREG] ; xreg[n]は、n番の汎用レジスタ

  ※Branch_and_Link機能付き, callにもなる

- jump_reg_indirect IREG1,REG2

  pc ← xreg[IREG1] + REG2

  ※Branch_and_Link機能付き, JALにもなる

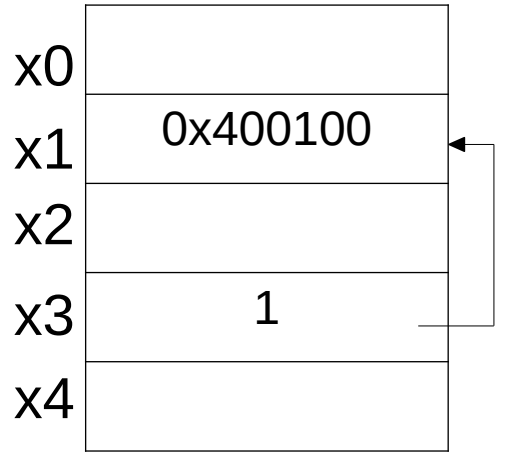
- mov_reg_regindirect dst,ireg

  dst ← xreg[ireg]

- mov_regindirect_reg ireg,src

  xreg[ireg]←src

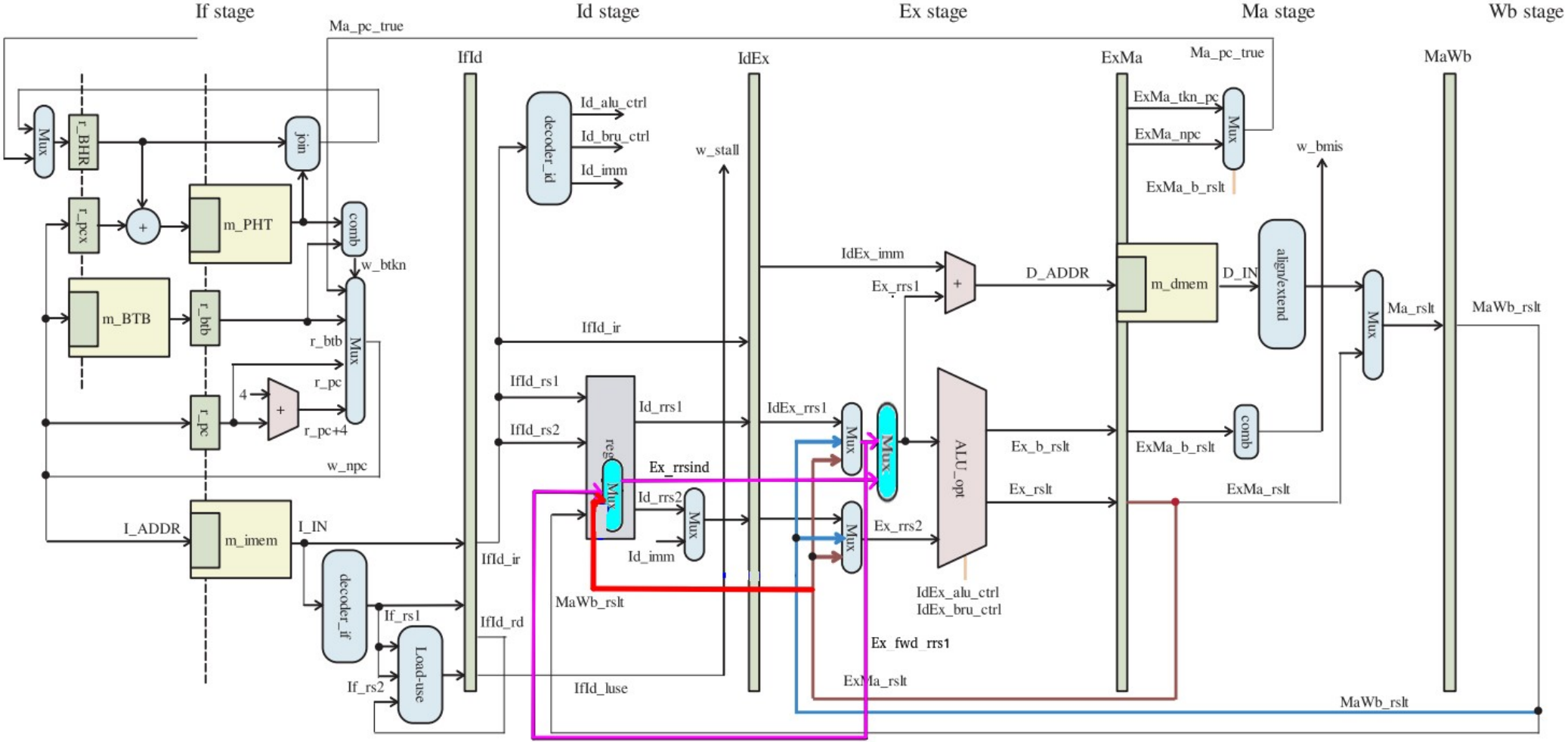
IREGがx3のとき
X1が指される。
参照の場合、実効値0x400100



mov_reg_regindirect

- `x[rd] = x[x[rs1]]`
- `I-type`
- フォワーディング部の修正
  - 次ページ参照

mov_reg_regindirect, branch_reg_indirect 追加後

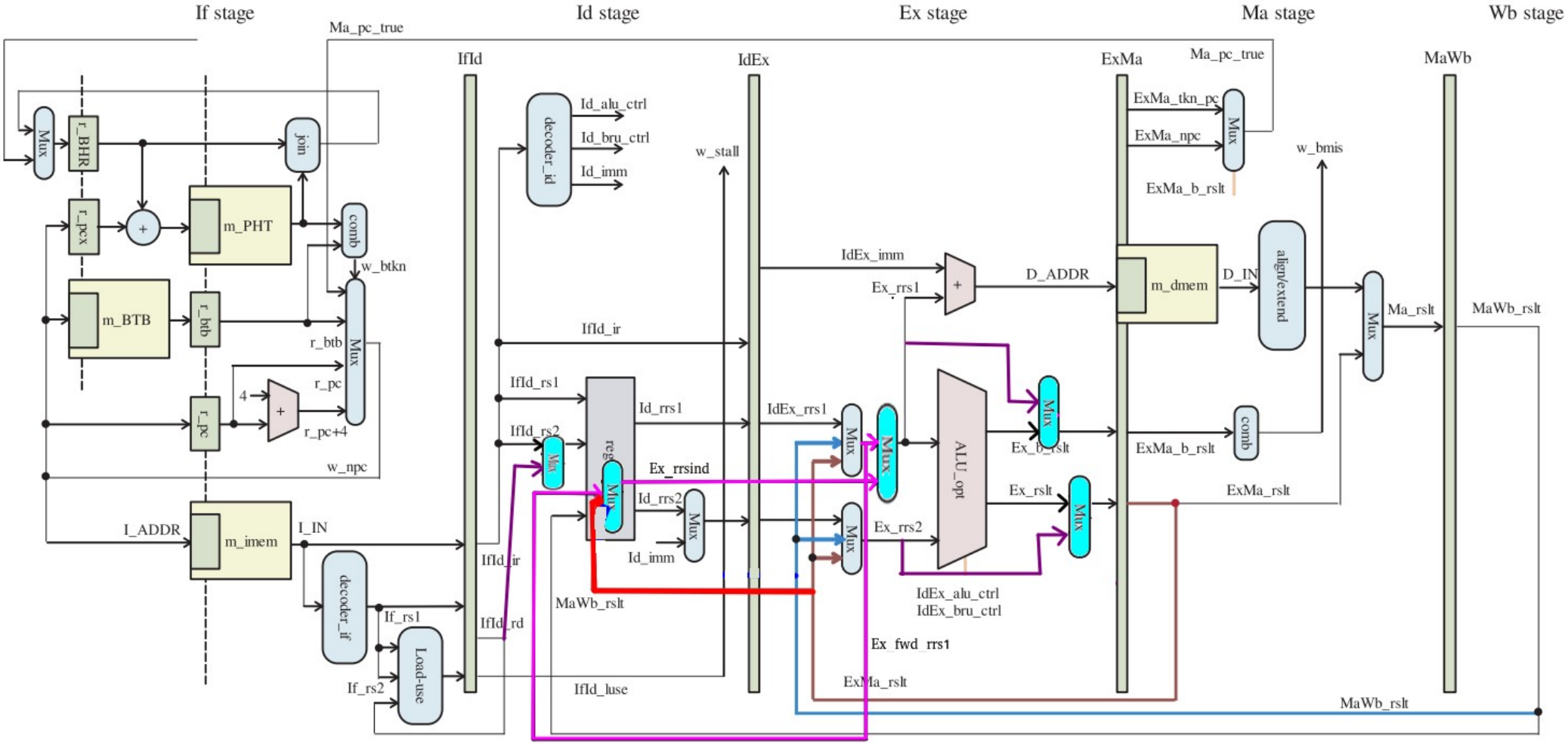


2022/JUL/15 a

mov_regindirect_reg

- x[x[rd]] = x[rs1]
- I-type
- 変更点
  - Id stageではrdをrs2として扱う
    - rs2のフォワーディング部を利用するため
  - Ex stageではフォワーディング後のrs2をExMa_rdに書き込む
  - 次ページ参照
- mov_reg_regindirectに追加で実装した
  - Fmaxのさらなる低下はなし

mov_reg_regindirect・mov_regindirect_reg追加後



2022/JUL/15 b

GNU Prologコンパイラ、新命令 対応ずみ

- gplc(GNU Prologコンパイラ)にオプション --emit-regind を新設
- 実測
  - FPGA(Nexys Video)に松竹CPUコア, IMEM=256KB, DMEM=1MB
  - Clock: 50MHz
  - ベア・メタル用runtimeルーチンを用意

  - (AX1001のメモリには載りきらなかったことには、触れない)

加速命令 使用の実測値

実行時間	加速命令 不使用(ms) N	加速命令 使用(ms) U	加速率(%) (N-U)/N*100	備考
queens.pl (16queen) (10回繰り返し)	61,638	58,930	4.39	みんな大好き 8queen
cal.pl (10回繰り返し)	11,026	10,827	1.80	万年カレンダ 数値計算
ham.pl (10回繰り返し)	52,591	52,324	0.51	???

とてもシンプルな機構(線を引っ張るだけ)

4%以上の加速が得られているので、十分 効いている

コスパ良し

シミュレーション & テスト･パターン 重要

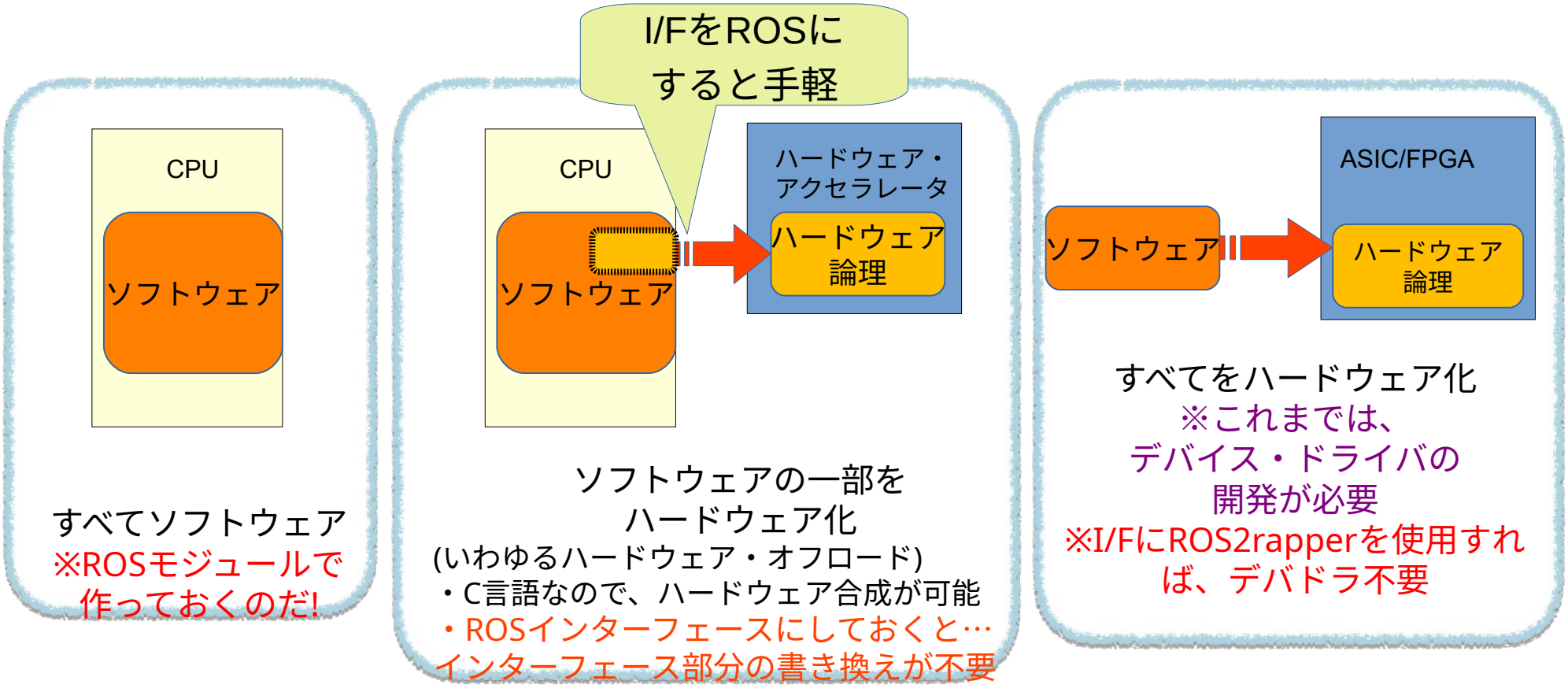
- シミュレータは良い
- 精度を高めると、シミュレーション遅い (^^;
- テスト・パターン
  - ネットワーク通信のテスト・パターン作成
    - 本当の通信をキャプチャ
    - FPGA 試作版があると、ありがたい

高位合成で
時代が変わった

そして
ROSを
汎用インターフェースに

高位合成によりHW⇔SWの行き来が自在

- C言語(HLS高位記述言語)で書くと、どんな形でも、どこでも実行できる
- ROS2インターフェースで柔軟 ⇔ これまでは、専用デバイス・ドライバとか必要



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、配布を準備中
  - LGPL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

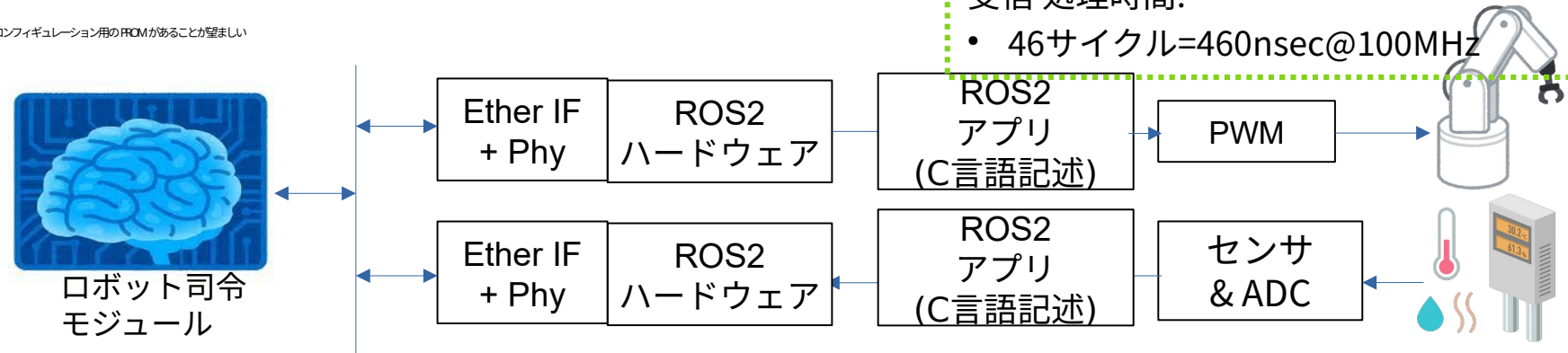
最大周波数: 121.01MHz

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

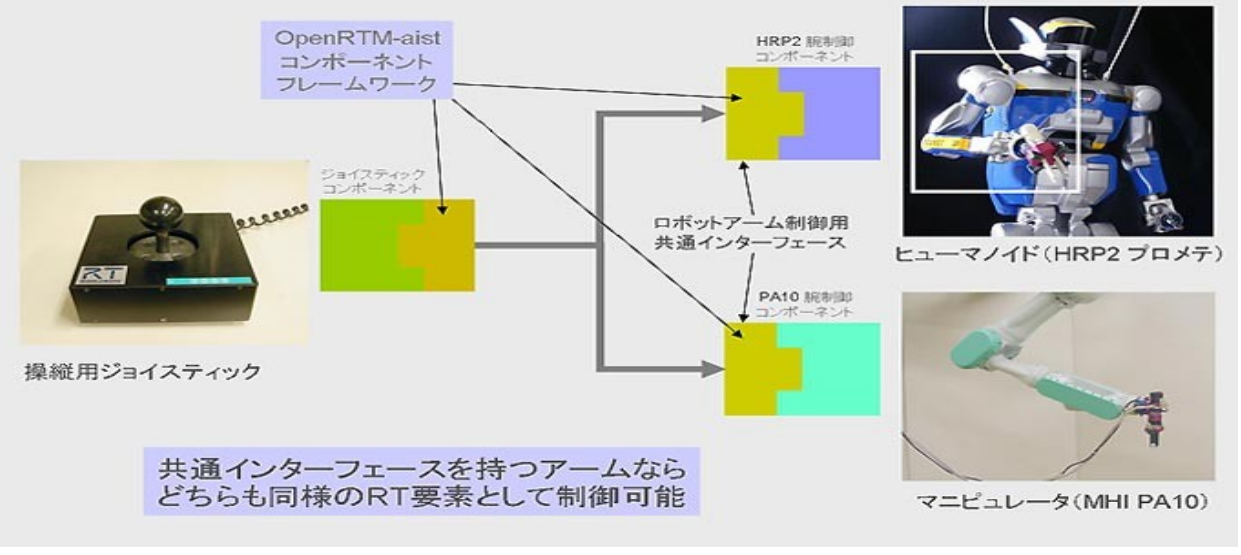
受信 処理時間:

- 46サイクル=460nsec@100MHz



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(RRI WG3)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
ロボット革命・産業IoT イニシアティブ協議会(RRI)
https://www.jmfrri.gr.jp/



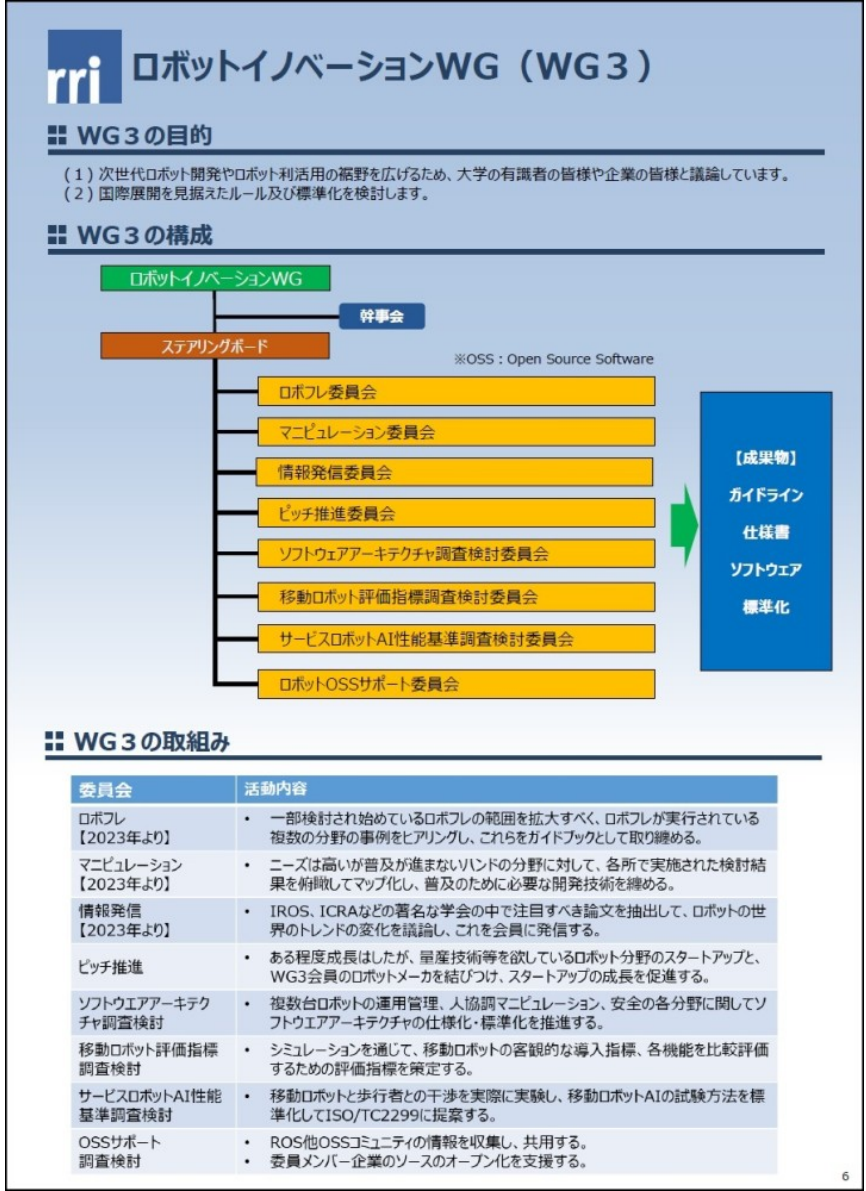
引用元 http://www.aist.go.jp/aist_j/press_release/pr2005/pr20050224/pr20050224.html

ロボット革命・産業IoT イニシアティブ協議会(RRI)

RRI https://www.jmfrri.gr.jp/

- WG3 ロボット OSSサポート委員会
- ROS (ROS2)振興

https://www.jmfrri.gr.jp/wp-content/uploads/2024/12/Pamphlet_WG3-721x1000.jpg



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

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
- LSI版は、超 低消費電力(電池駆動 可能)

既存ROS2の難点

- (事実上)Linuxが必要
  - 電気を喰う
  - MMU付きCPU
  - メモリ たっぷり
- ウイルスが付くかも…
  - ロボット系の人は、ウイルスを警戒

セキュリティ面でのオール・ハードウェアの優位性

- セキュリティ・ホールができにくい
  - スタック・オーバフロー 攻撃は、原則起きない
    - 「制御を取られる」ということが無い
      - CPUでは、悪者コードの先頭へ、 プログラム・カウンタをセットする
  - ユーザの書いたアプリケーションが、セキュリティ・ホールを産むことはありえる
    - 下回りが完璧でも、ユーザのポカはあり得る
- DOSアタック (無駄なパケットを間断なく投げつけてくる) には、勝てない(負けるとも言わないが)
- 仮にセキュリティ・ホールを突かれても、リセットすると 完全復帰
  - FPGAのコンフィギュレーションROMを書き換えることは、ほぼ不可能
    - ユーザ・アプリケーションがバカでも、そこまではいかないだろう…(?)
  - ※Linuxなどの複雑なOSは、重要な設定ファイルなどを書き換えられてしまったら、再起動してもダメ

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

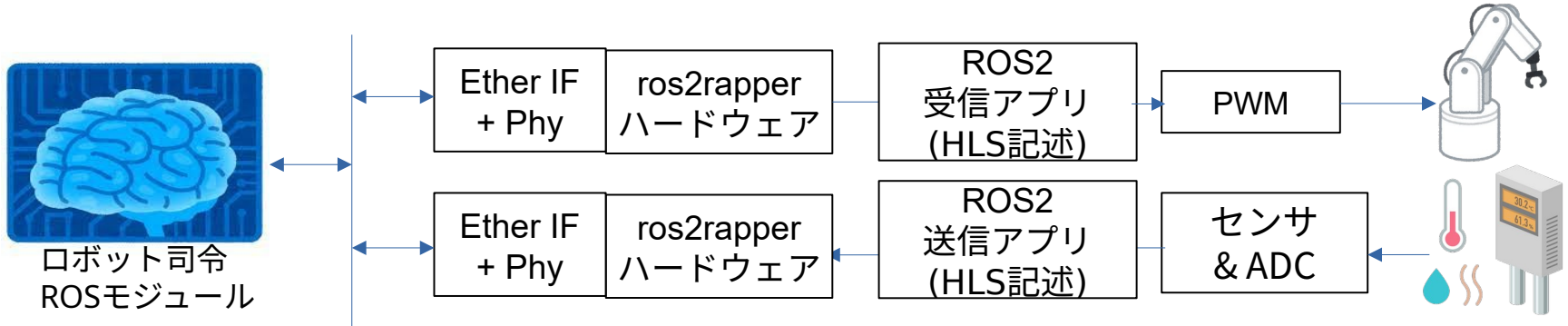
FPGA使用資源

- LUT: 33666
- FF: 13087

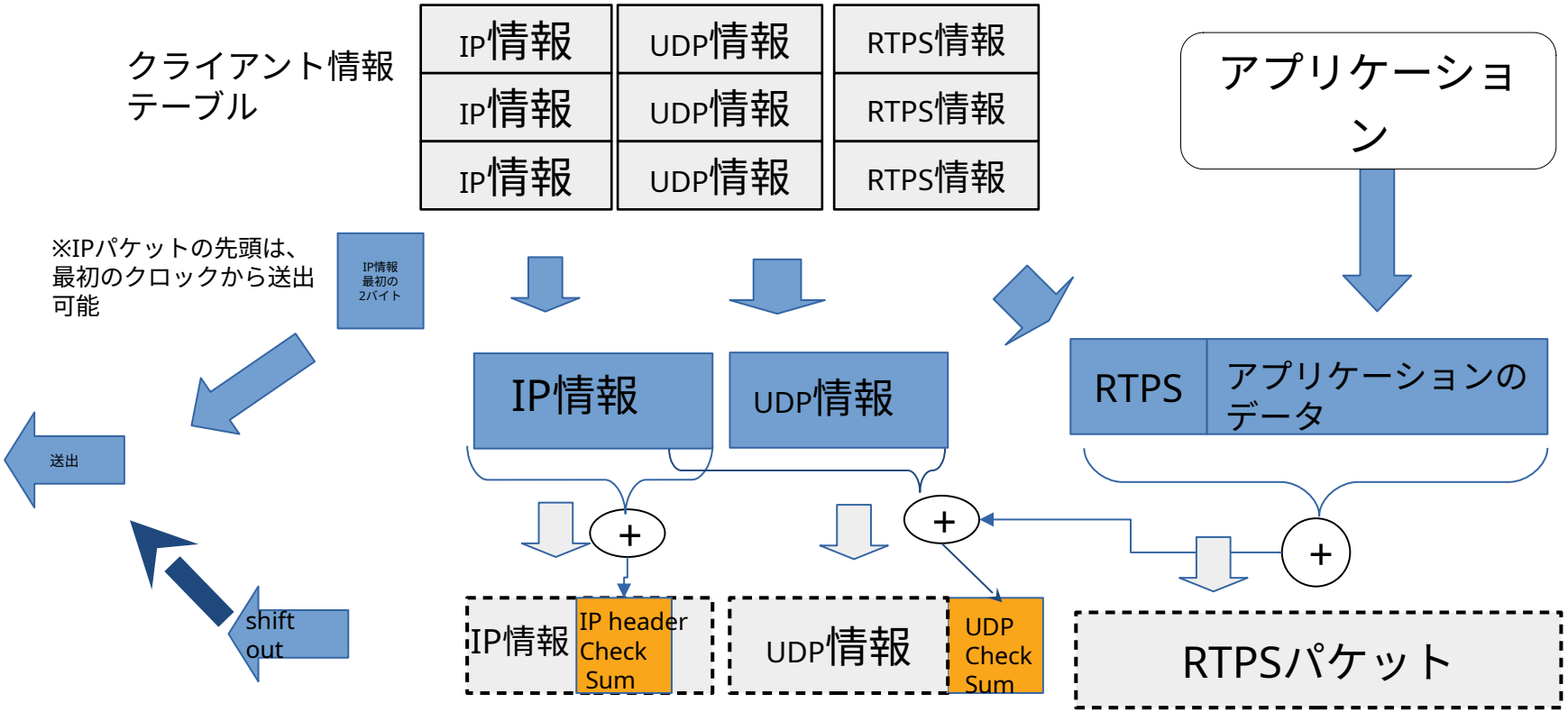
最大周波数: 121.01MHz

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



ネットワーク・パケットの並列生成(2021/SEP/27)



・FFで、シフトレジスタを構成
-FFなので 並列read可能
-FFなので、並列プリセット可能

※すべての情報が確定していれば、送信パケット生成は1クロック

・IPヘッダの最初の2バイトは、早期に、送出可能

Laxer AX1001 消費電力概要

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

CPU無しで
動作可能

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

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

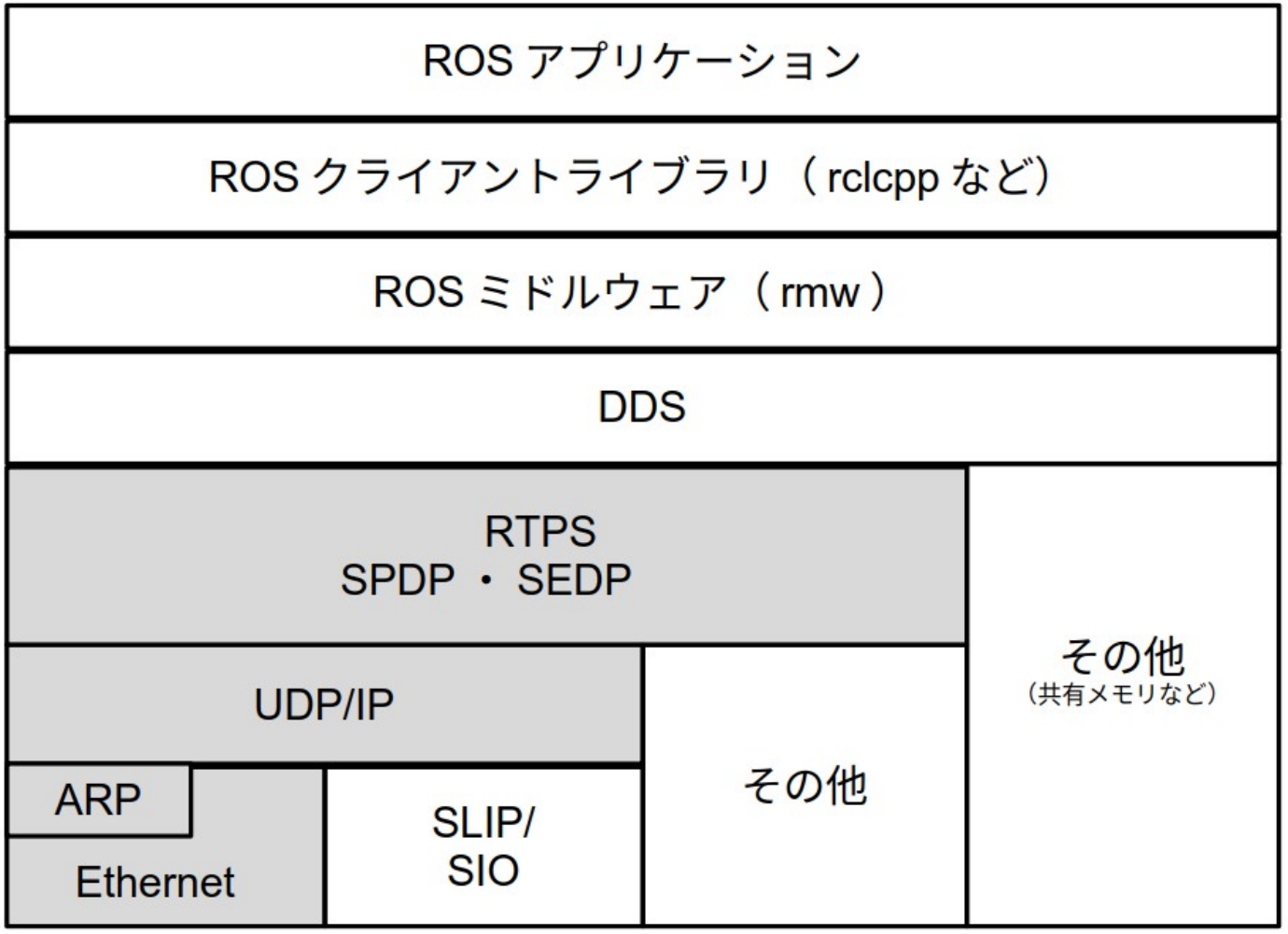


Fig.1.1 ROS2 アーキテクチャにおける ROS2rapper の位置

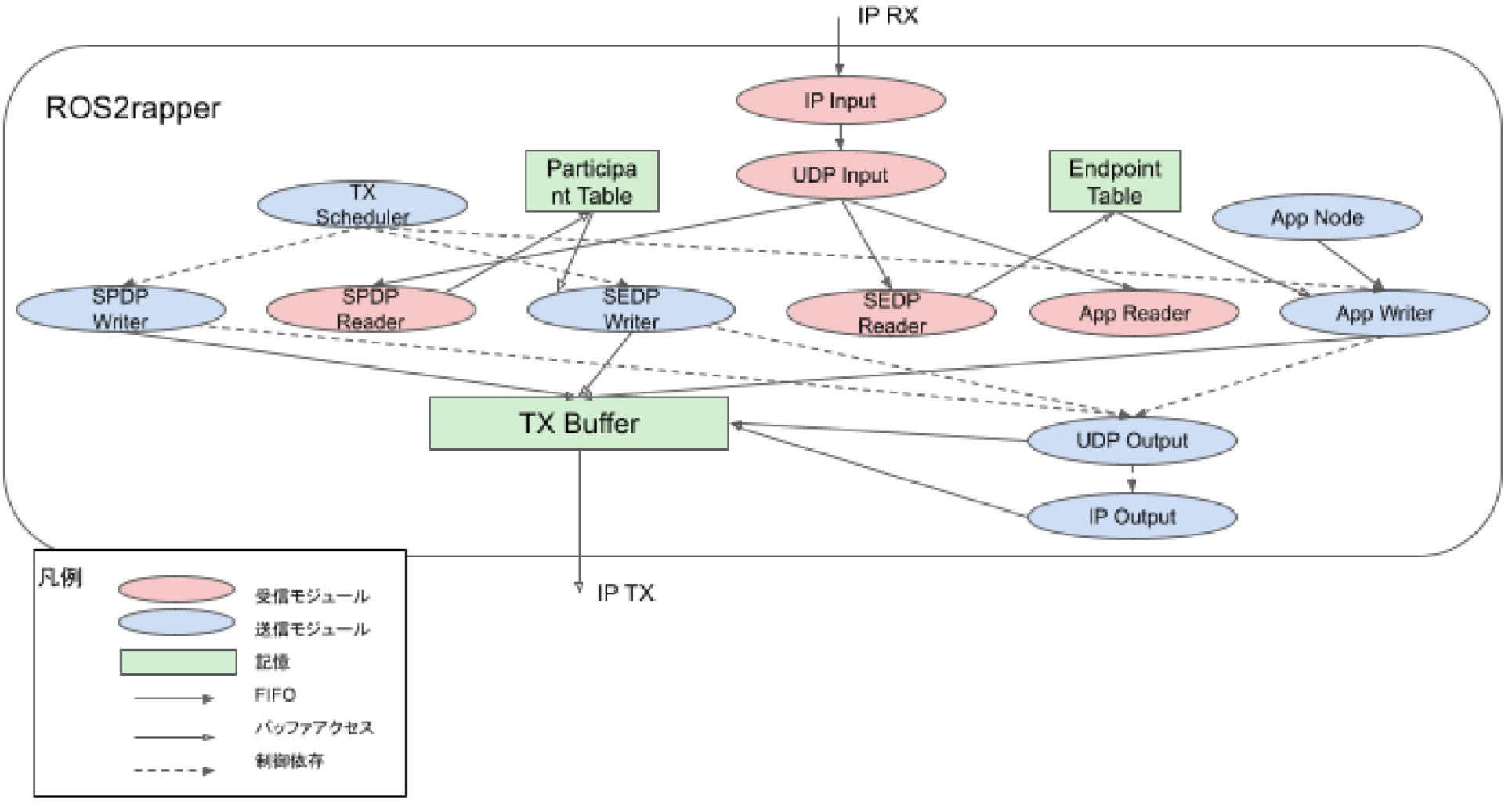
ROS2プロトコル

- 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)



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

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）	**54,223**
Raspberry Pi 3 ---> x86_64マシン（Ethernet UDP通信）	**6,047**
（参考）x86_64同一マシン上のプロセス間 （共有メモリ通信）	**14,556**

(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 - 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）	**12,201**
x86_64マシン ---> Raspberry Pi 3（Ethernet UDP通信）	**5,635**
（参考）x86_64同一マシン上のプロセス間 （共有メモリ通信）	**13,973**

(Messages/sec)

測定結果（10回の平均値）

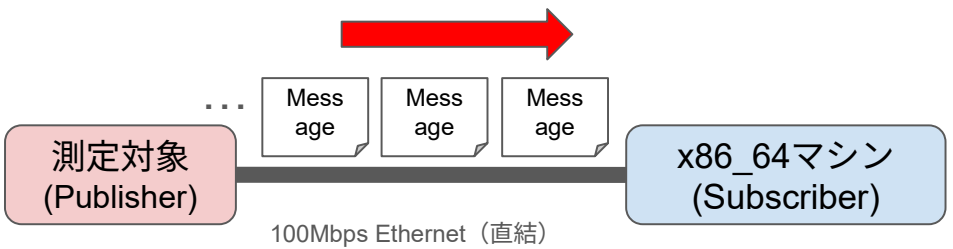
ROS2メッセージ送信速度比較の環境

- **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



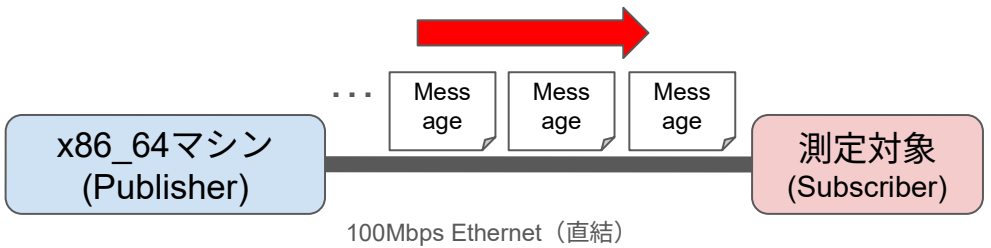
ROS2メッセージ受信速度比較の環境

- **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

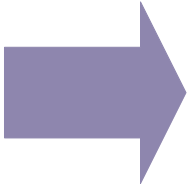


CPU脳を
たたき直す

もう、**CPUは**(簡単には)**速くならない**よ

- ついに、微細化 限界
- 微細化 による
  - 高集積
  - 高クロック周波数

  は**終了**



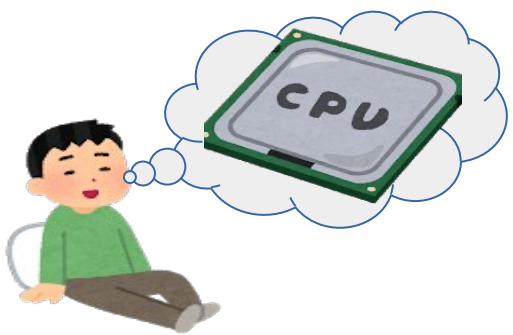
- 専用 ロジック回路による
  - 高速化
  - 省 消費電力

  を行うしか

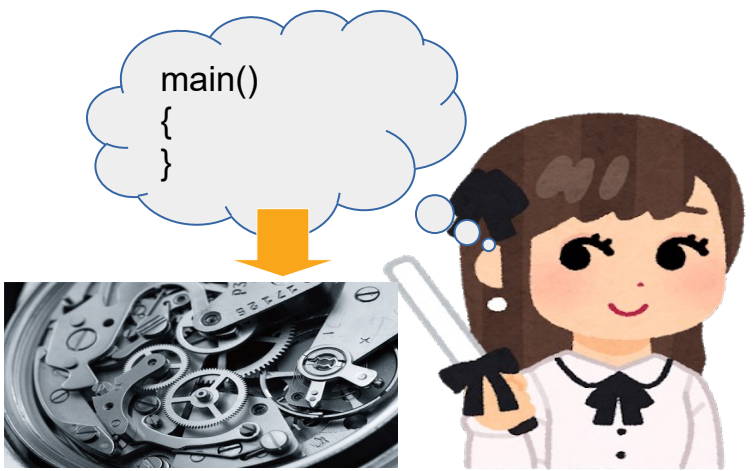
自由ASIC時代

- 「CPU脳」な人類を、パラダイムシフト
  - CPU抜き アーキテクチャを、すすめる

- CPU抜き システムのアーキテクチャ決定をサポートすべし



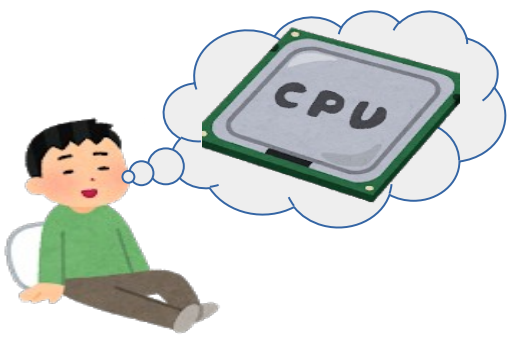
旧人類 CPU脳



ハードウェア
(状態数が たいへんに大きい精密機械)

脱CPUアーキテクチャ を 推進する

- 新しい細粒度 高並列アーキテクチャの時代
  - 1port RAMをやめさせる、D-FFを使わせる
- 同時並列にバラバラにデータ・アクセスできる



CPU脳プログラマ

ツールと教育

背景:
カスタムLSI
設計&製造の
民主化

日本の半導体 産業 復興!

- 国内8社が半導体製造会社「Rapidus」設立
- 経産省キモ入り　「10年の遅れ」取り戻す
- キオクシア、ソニーグループ、ソフトバンク、

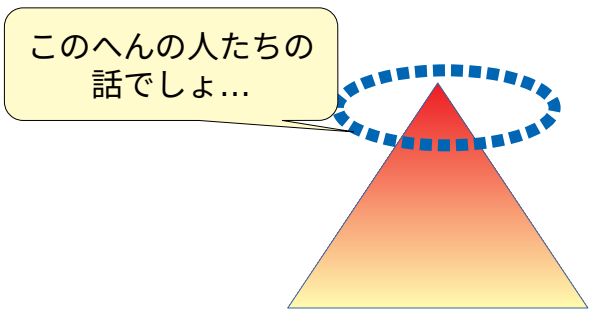
デンソー、トヨタ自動車、NEC、NTTが

それぞれ10億円、三菱UFJ銀行が3億円を

出資した。

https://www3.nhk.or.jp/news/html/20221111/k10013887921000.html

- 半導体 工場は、かろうじて 最新のものがある
- でも、「お高いんでしょう〜?」
- 技術者 不足
  - 半導体 設計 技術者
  - 論理回路 設計 技術者



技術者 不足をStop!日本の半導体 産業 復興!

·OSSの開発ツールで、LSI 開発

- 無料ツールの使い手が増える → 技術者不足 解消!
  - 半導体 設計 技術者
  - 論理回路 設計 技術者

·半導体 工場は、「お高くない」ものもある

- 65nm とか、安くて かなりいい

·レガシーファブの活性化

産総研 直下のAIST Solutionsも、ロングテール半導体開発を推進

Open-Source は、ロングテール半導体開発の夢を見るか？

- Do Open-Source Dream of Long-Tail Semiconductors? –

11.17(金) 15:30-16:10 | 展示会場内　Room E

AIST Solutions
半導体事業 プロデューサー　岡村 淳一

AISol の半導体事業の目標

国内の半導体アセット（チップ製造能力）を
本プロジェクトのプラットフォームに再整備することで
専用半導体の設計の参入障壁を下げ、
国内産業が専用半導体にて国際競争を勝ち抜く環境を提供する。

AIST SOLUTIONS

Open Source Silicon for Japan

OpenSUSI,AIST Solの団体

- AIST Solutionsは、2024年4月5日付で一般社団法人OpenSUSIを設立しました。

- https://www.semiconportal.com/archive/editorial/industry/240423-aistsolutions.html より引用

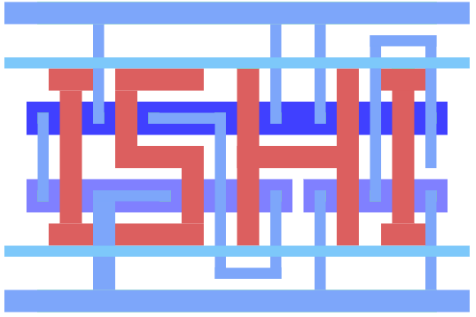
- 産業技術総合研究所が日本版「半導体ICの民主化」プロジェクトを進めていることがわかった。産総研の総責任者である理事長の石村和彦氏（図1）は、産総研が開発した技術を社会実装して世の中の役に立たせようという石村改革を就任以来進めてきた。2023年設立したAIST Solutionsは社会実装の先頭部隊。2024年4月には半導体ICの開発に向けて「OpenSUSI」を設立した。これこそ半導体の民主化を狙った組織である。

- 4月に設立されたOpenSUSIが扱うのは、マーケットが豊富なレガシープロセスの半導体であり、ロングテール製品向けにASIC（アナログやデジタル回路を中心に設計されたハードウエアIC）やSoC（CPUなどのプロセッサを中心にソフトウエアも含めたIC）などを開発するためのサポートを行う。

- レガシープロセス:最先端の配線構造やトランジスタ構造をフルに3次元活用しない。

ISHI会

- https://ishi-kai.org/
- 「石 (チップ)」の会
- OSS EDAを実際に使用して、IPを開発する人たち
- 学識経験者/大学の先生も多数
- ワールドワイドなプロジェクトを実施中
  - 日本の担当は、オンチップ 安定化電源
- 初心者 指導も熱心に行っている
  - JASA RISC-V WGにも出席



超ローコストで
半導体(LSI) 開発が
可能に!

Googleがカスタム半導体の民主化・自由化を推進

- Googleと半導体ファウンドリの「SkyWater」が協力し、業界初となるオープンソースのPDKを公開
  - Skywaterは2017年に米Cypress Semiconductorからスピンオフしたファウンドリ企業

- PDK プロセス設計キット
  - ある特定の半導体プロセスで回路設計を行う際に必要な設計情報
  - トランジスタ配置の制約条件などが書かれている
- 半導体の設計者は、半導体製造のファウンドリから「Process Design Kit(PDK)」と呼ばれる開発キットを購入
- 半導体ファウンドリが提供するPDKは高価 → それが**無料 OSSに!**

- SkyWaterの130nmプロセス「SKY130」で半導体チップの製造を行うための設計を無料で行うことが可能

- GitHub - google/skywater-pdk: Open source process design kit for usage with SkyWater Technology Foundry's 130nm node.

https://github.com/google/skywater-pdk

DARPAによるOpenROADへの$200Mの投資

- オープンソースEDAソフトウェア
  - 数十年前から存在。最先端ノード製造工程の複雑で商業ツールに遅れた
  - EDAツールのコスト：3つのEDA会社が25%を研究開発に投資
    - Nvidia、Apple、Qualcommなどエンタープライズ顧客
    - ＄10のIoTチップを作る小さなチーム
- 20の既存ツール → 1アプリにバンドル → エンドツーエンドのフロー
  - Qflow： Verilogソース → 物理レイアウト
  - SPICE： 回路シミュレーションプログラム
  - Magic： VLSI回路レイアウトの作成と変更
    - Mead-Conway設計手法で、単純ルールで、基本セルを階層設計
  - VIS：Verification Interacting with Synthesis

川崎俊平氏の資料より引用
https://riscv.or.jp/wp-content/uploads/RVDs2022Spring_SHC-2022-06-02_c.pdf

eFabless 事例1

- NEDO資金も受け、日本人 河崎氏も、実際に、LSIを開発した。

  ※河崎氏は、RISC-V Foundation ボードメンバで、JASA RISC-V WGメンバでもある

- 「Google社 が スポンサとなりeFabless社 の オープンソースシャトル を 使用し30日でRISC-V半導体 を 設計試作」
  - https://riscv.or.jp/2022/05/marmot-risc-v-asic/

eFabless 事例2

- 今村 謙之氏も、実際に、LSIを開発した。
- 「Kernel/VMレイヤーを自分色に染める！By ISHI会」
  - https://www.slideshare.net/noritsuna/kernelvmby-ishi
- イベント「はじめての半導体チップ設計」のアーカイブ動画を公開
- https://ishi-kai.org/event/report/2023/08/25/AugustEvent_0804_Report.html

https://www.youtube.com/watch?v=W-HDNI4IK5A

悲報:eFabless 倒産

- eFabless倒産…
- だが、まだ行ける

eFabless居なくても平気だぜ!

Tiny Tapeout

- https://tinytapeout.com/
- ISHI会 今村氏が熱心に普及活動

OpenSUSI

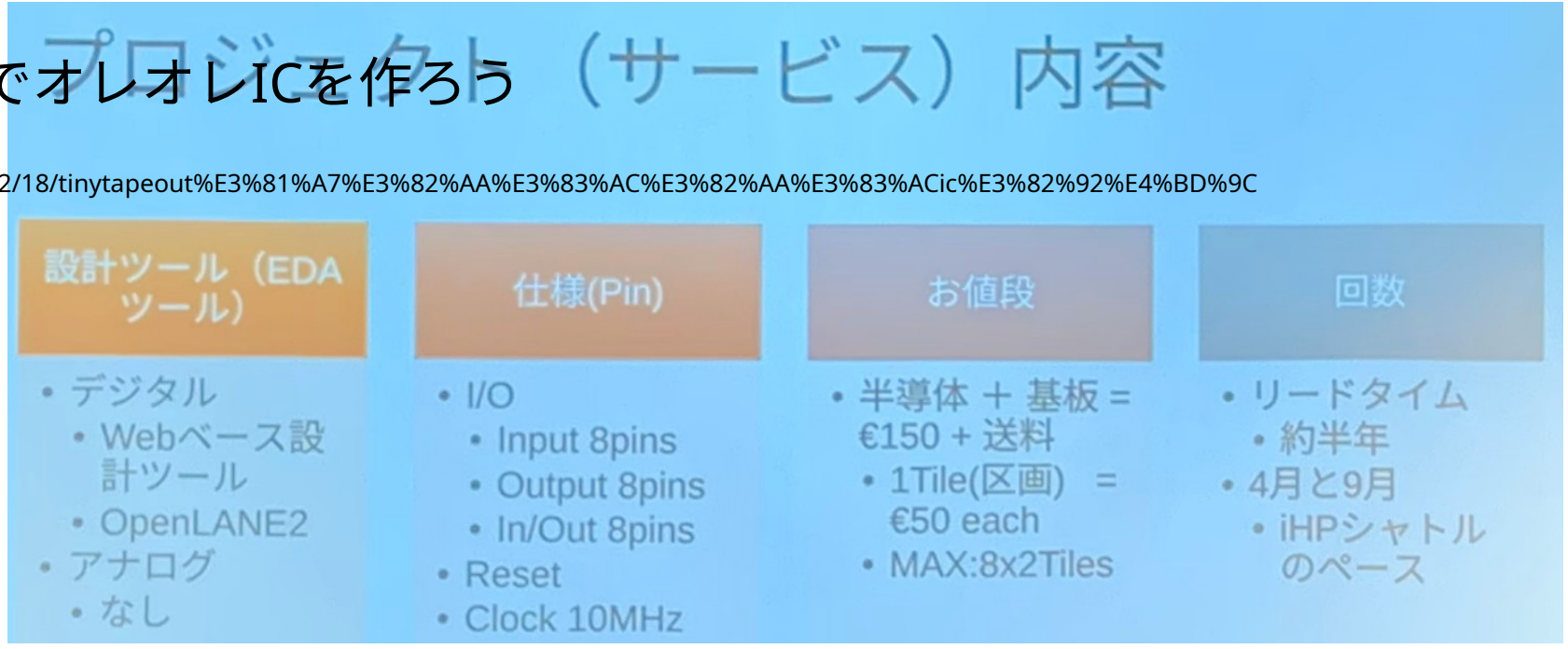
- OpenSUSIは、国内の工場(レガシーfab)で、オレオレLSIの開発を推進

Tiny Tapeout

- https://tinytapeout.com/
- ISHI会 今村氏が熱心に普及活動
- 半導体製造(TinyTapeout)に挑戦しよう！
  - https://www.slideshare.net/slideshow/tinytapeout/260745768
- TinyTapeoutでオレオレICを作ろう
  - https://t-techlab.com/2024/02/18/tinytapeout%E3%81%A7%E3%82%AA%E3%83%AC%E3%82%AA%E3%83%ACic%E3%82%92%E4%BD%9C%E3%82%8D%E3%81%86/

学生などの若者たちも挑戦中

JASA RISC-V WGもTinyTapeout活動を(精神的)支援



ミニマルファブ

一般社団法人 ミニマルファブ推進機構

MINIMAL(Minimal Fab Promoting Organization)は、

半導体、MEMSなどマイクロデバイスの多品種少量生産を可能とする革新的な

産業システム(ミニマルファブ)の発展と普及を支援する世界唯一の団体です。

https://www.minimalfab.com/

- 半導体 1個を手作りで作れる
  - 手間は掛かるが、費用は超安い

https://www.semiconportal.com/archive/editorial/conference/report/130705-minimalfab.html?print
より引用

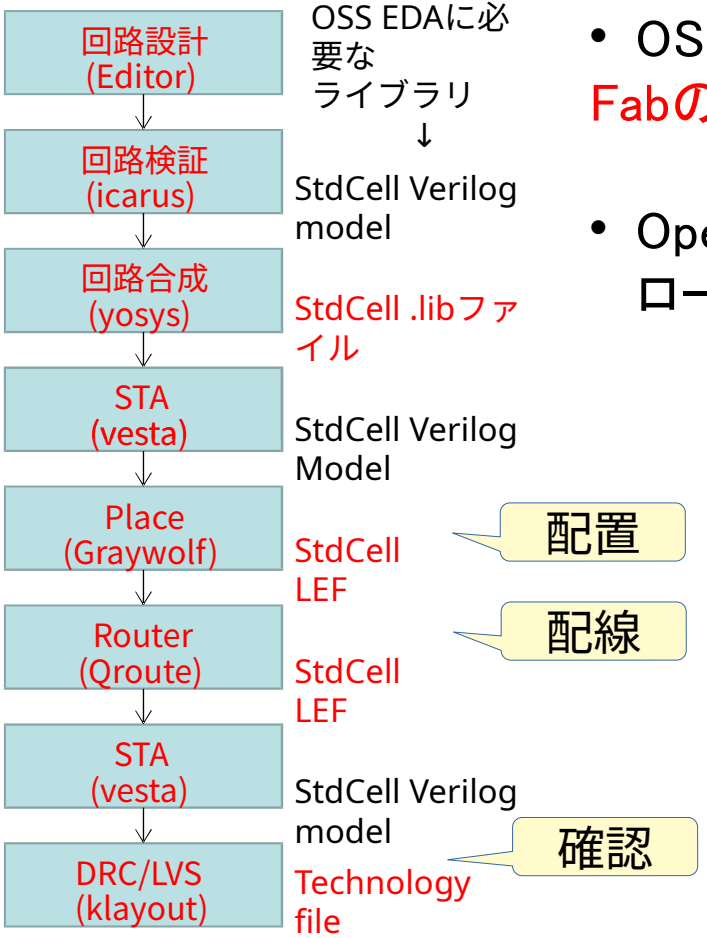
半導体(LSI) 開発が
OSSで自由な世界に

半導体チップの設計フロー3つの要素

- RTLデザイン　〇
- EDAツール 〇
- デジタル合成フロー〇
- PDKデータ　← これがOSSに

OSSによるLSI開発のEDAフロー

ロジック部開発フロー



- OSS EDAに必要なライブラリは Fabのデータから変換が必要
- OpenRAM、PLL、アナログは別なフローになる

デジタル合成フロー(Digital Synthesis Flow)

- Digital synthesis flowは、ツールと手法
  - RTL → 物理回路 を合成
  - FPGA ならば、Xilinx, Intel などのコンフィギュレーション・コード
  - 特定の半導体工場(ファブ)で作るIC(LIS)の場合は、ファブのプロセス・テクノロジでのレイアウト
    - PDK情報が必要
- これまで
- 半導体用のデジタル合成フローは、

ケーデンスやシノプシス

という大手企業のみが供給していた

- FPGA 用は、Xilinx, Intel などFPGAメーカがツールを提供

  OSSではないが、無料で配られていることも多い

Graywolf OSSデジタル合成フロー・ツール

- トランジスタ配置ツール
  - 主にQflowと併用
- TimberWolf 6.3.5からフォークした
  - TimberWolf はイェール大学で開発され、商用化されるまで、しばらくはオープンソースとして配布された
  - TimberWolf のオープンソース版の最後のバージョンは詳細ルーティングを実行しない
    - しかし、プロ仕様の配置ツールだった
  - graywolf の主な改善点
    - ビルド プロセスがより合理化された
    - 通常の Linux ツールとして動作する
      - 最初に環境変数を設定することなく
      - どこからでも呼び出すことができる

https://github.com/rubund/graywolf

Qflow OSSデジタル合成フロー・ツール

※Graywolfと併用するのがよい

- デジタル合成フローは、verilog や VHDL などの高水準言語で書かれた回路設計を物理回路に変換するために使用される一連のツールとメソッドです。
  - Qflow 1.3: An Open-Source Digital Synthesis Flow
  - http://opencircuitdesign.com/qflow/welcome.html
  - OpenCores内 情報
    - https://opencores.org/howto/eda
  - Icarus Verilog Simulator:Verilog simulation and synthesis tool
  - Verilator: free Verilog HDL simulator
  - GHDL VHDL simulator

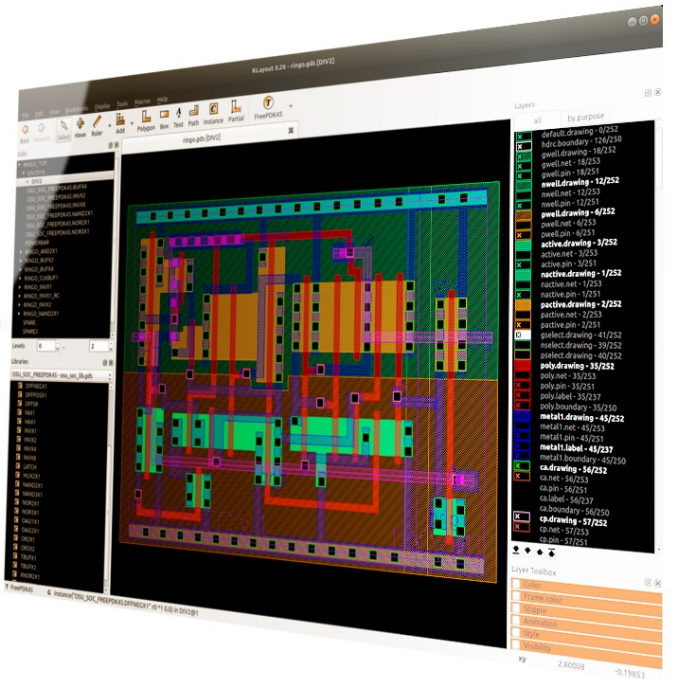
OpenRAM: RAM合成ツール

- OpenRAMは、RAMを合成する
  - 同時ではない、read/write のフツーのRAMは合成できる
- OpenRAMは、同時1read & 1write のRAMが仕様上は合成できるはず。
  - だが、ダメ(残念)

Klayout

- マスクの確認
- 分析
- https://www.klayout.
-

```
# The PCell declaration for the circle
class StarPCell < PCellDeclarationHelper

  include RBA

  def initialize

    # Important: initialize the super class
    super

    # declare the parameters
    param(:l, TypeLayer, "Layer", :default => LayerInfo::new(1, 0))
    param(:r1, TypeDouble, "Inner radius", :default => 1, :unit => "µm")
    param(:r2, TypeDouble, "Outer radius", :default => 5, :unit => "µm")
    param(:n, TypeInt, "Number of rays", :default => 32)
    param(:da, TypeInt, "Ray angle", :default => 5, :unit => "deg")

  end

  def display_text_impl
    # Provide a descriptive text for the cell
    "StarPCell(L=#{l.to_s},R1=#{'%.3f' % r1.to_f},R2=#{'%.3f' % r2.to_f},N=#{n})"
  end

  def produce_impl

    # This is the main part of the implementation: create the layout

    # compute the ray parts and produce the polygons
    d = Math::PI * da * 0.5 / 180.0
    a = 0.0
    n.times do |i|
      dpts = [
        DPoint::new(r1 * Math.cos(a - d), r1 * Math.sin(a - d)),
        DPoint::new(r1 * Math.cos(a + d), r1 * Math.sin(a + d)),
        DPoint::new(r2 * Math.cos(a + d), r2 * Math.sin(a + d)),
        DPoint::new(r2 * Math.cos(a - d), r2 * Math.sin(a - d))
      ]
      cell.shapes(l_layer).insert(DPolygon.new(dpts))
      a += Math::PI * 2 / n
    end

  end

end
```



GAFAなどが自社向け半導体の開発に力を入れる理由

- データセンタの消費電力 削減
  - 自社製 専用半導体で電力削減
- 設計技術だけで自社半導体はできる
  - 半導体工場レス
  - RTL(デジタル論理)設計ができれば
    - 専用アルゴリズム用LSI
- オーダーメイドICは、FPGAでは不満
  - FPGAは消費電力 大
  - FPGAは遅い

国内

- 政府が、LSI産業 再興
  - JASAにも、経産省から LSI 開発者 教育についてヒアリングが来て、私もJASA技術本部長として応えました。
- LSI開発者の裾野を広げたい
  - 産業技術総合研究所なども、OpenEDAに注目

国内もLSI開発の民主化 推進

日本も、国の金で 施設を用意

ふくおかIST(公益財団法人 福岡県産業・科学技術振興財団)
福岡システムLSI総合開発センター
「システムＬＳＩ設計試作センター」

- http://www.ist.or.jp/lsi/pg04_02.html

- ベンチャー企業が半導体の設計ツールを無料(安価)で利用できる
- LSI設計、 少量 試作できる
  - 50～100万円 あれば、LSIの少量生産ができる仕組みがある

小規模EDA開発 日本でも流行

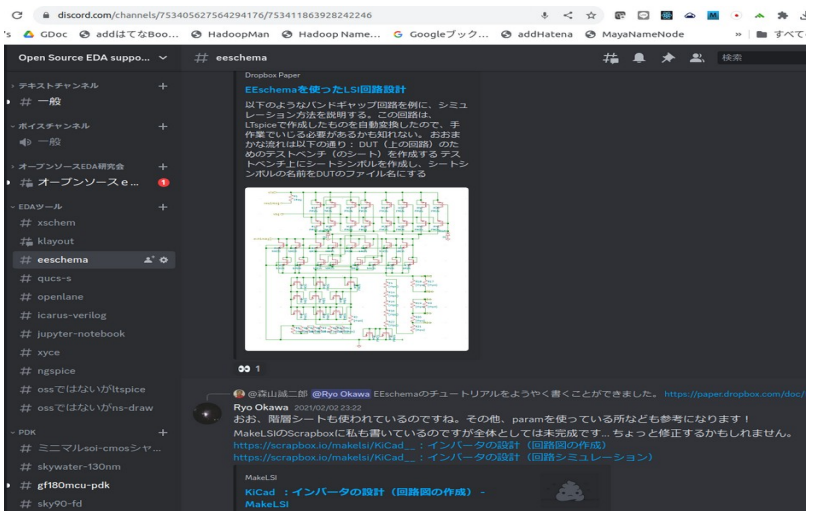
福岡システムLSI総合開発センター
「システムＬＳＩ設計試作センター」
の設計ツール一覧

EDA機能		製品名
ハイレベル設計	Cレベル合成	* CyberWorkBench ※NECの商品
フロントエンド設計	論理シミュレータ	* Incisive Enterprise Simulator L
	回路図エントリ	* Schematic Editor
		* ASCA
		* ASCA Basic
	シミュレーションIF	* Virtuoso ADE
		* ASCA Sim.faceA
	総合回路設計	* C^3
	Composer IFオプション	* Composer IF
	Verilog Interfaceオプション	* Verilog Interface
	SPICE Interfaceオプション	* Analog HSPICE IF
	アナログ回路シミュレータ	* Spectre circuit Sim
		* Msim
	汎用回路波形解析	* SimVision
レイアウト	レイアウトエディタ	* Virtuoso LE
		* ISMO
	Cadence Linkオプション	* Cadence Link（DF II Upgrade）
レイアウト検証 その他	DRC	* Calibre DRC
	LVS	* Calibre LVS
	IFオプション	* Calibre RVE
	DRC/ERC	* iDRC/ERC
	Caliber IFオプション	* Calibre IF
	寄生パラメータ抽出	* Calibre xRC

Open Source EDA Supporters (discode)

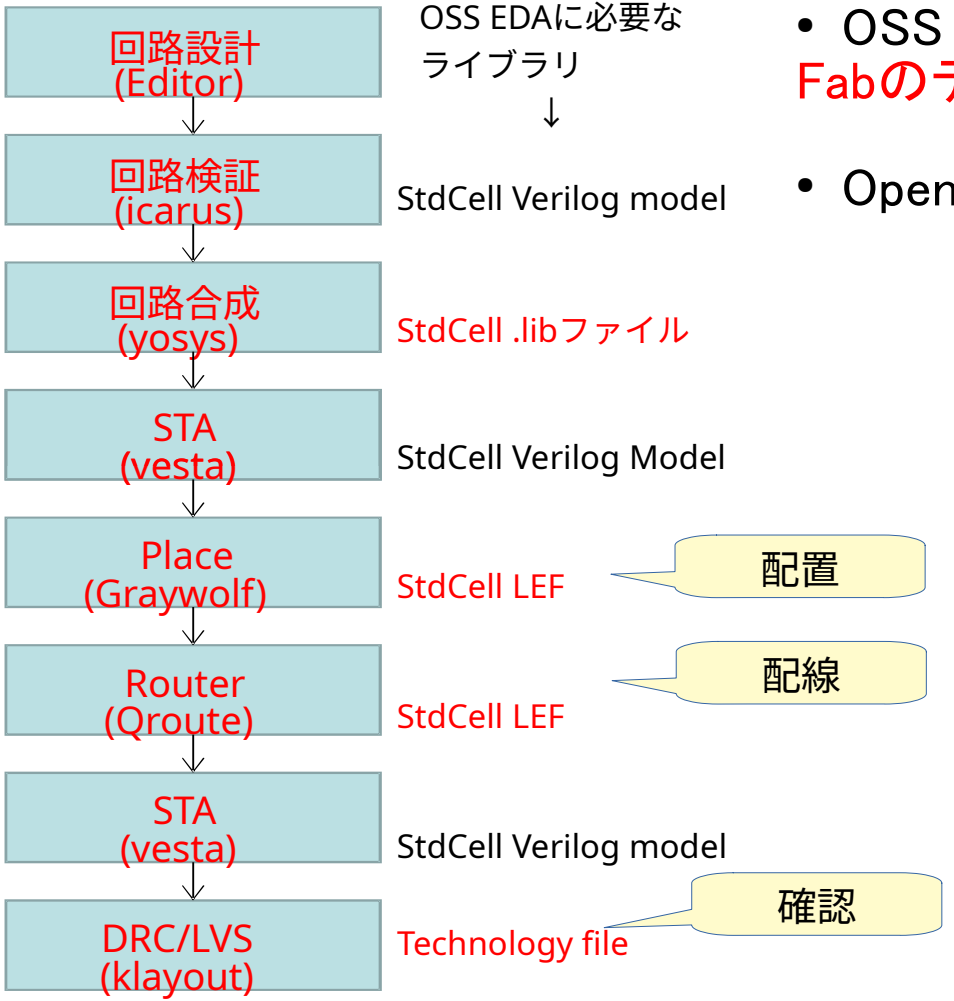
https://discord.com/channels/753405627564294176/932209975558832128

- OSS EDAをサポートする人たちの集まり
  - 親切で優しい
- ふくおかIST のOSS EDAサーバのユーザ会
  - サーバを維持するモチベーション
- オレオレ Open PDKを開発したり: 森山氏
- オレオレ Standard Cellライブラリを開発できるツールの開発者が居たり:西澤氏@早稲田大
- オープンソースＥＤＡフォーラム @福岡
  - 2023/DEC/08
  - OSSコンソーシアムEDA部会が共催
  - JASA OSS活用WG協力
- OSC福岡にも出展
  - OSSコンソーシアム&協力:JASA OSS-WG体制
  - 2023/DEC/09



OSSによるLSI開発のEDAフロー

ロジック部開発フロー



- OSS EDAに必要なライブラリは
Fabのデータから変換が必要
- OpenRAM、PLL、アナログは別なフローになる

俺のCPUのもと(?)

SPARCもオープンソースなソフトコアあり

- Open Sparc

https://www.oracle.com/servers/technologies/opensparc-overview.html

- LEONシリーズ
  - 欧州宇宙機関(ESA)が積極開発
  - Open Sparc の継続

https://en.wikipedia.org/wiki/LEON

  - Len3, 3FT,4,5
    - LEON3FT : Fault-tolerant processor

https://www.gaisler.com/index.php/products/ipcores
https://www.gaisler.com/index.php/products/processors/leon3

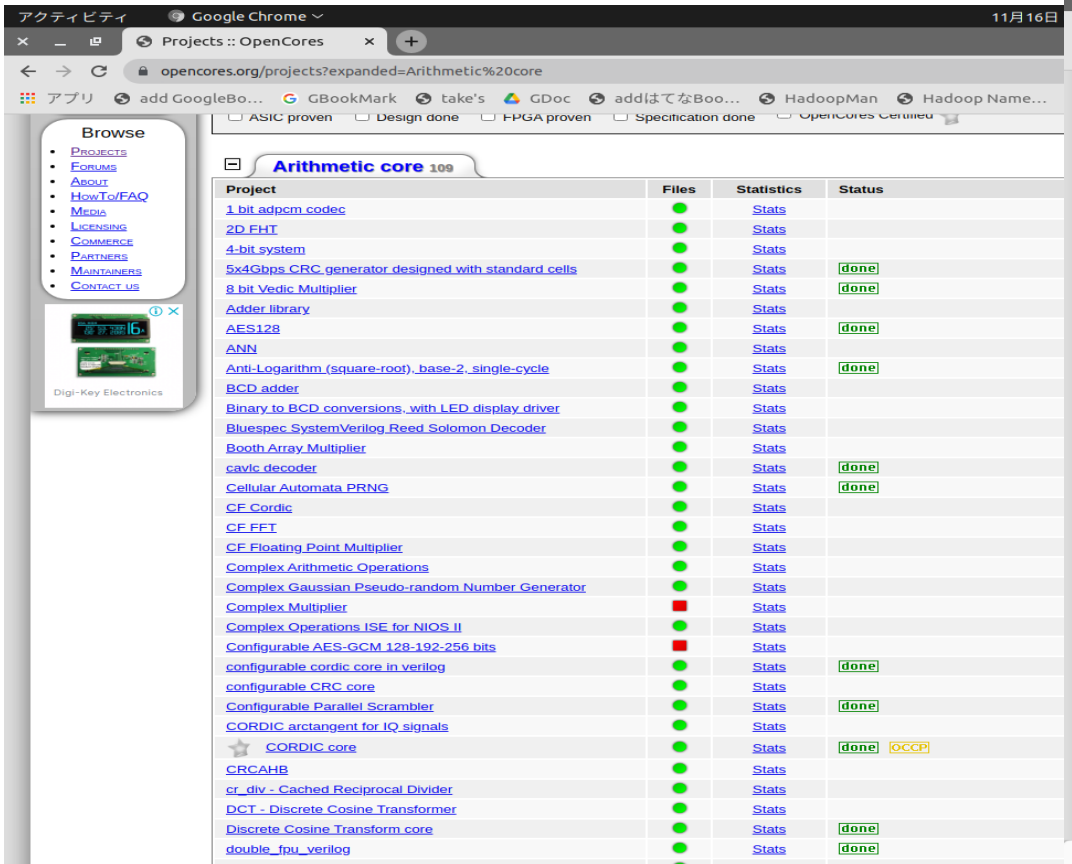
- Leon3 はGPL
- SPARC v8 が FPGAでも動作

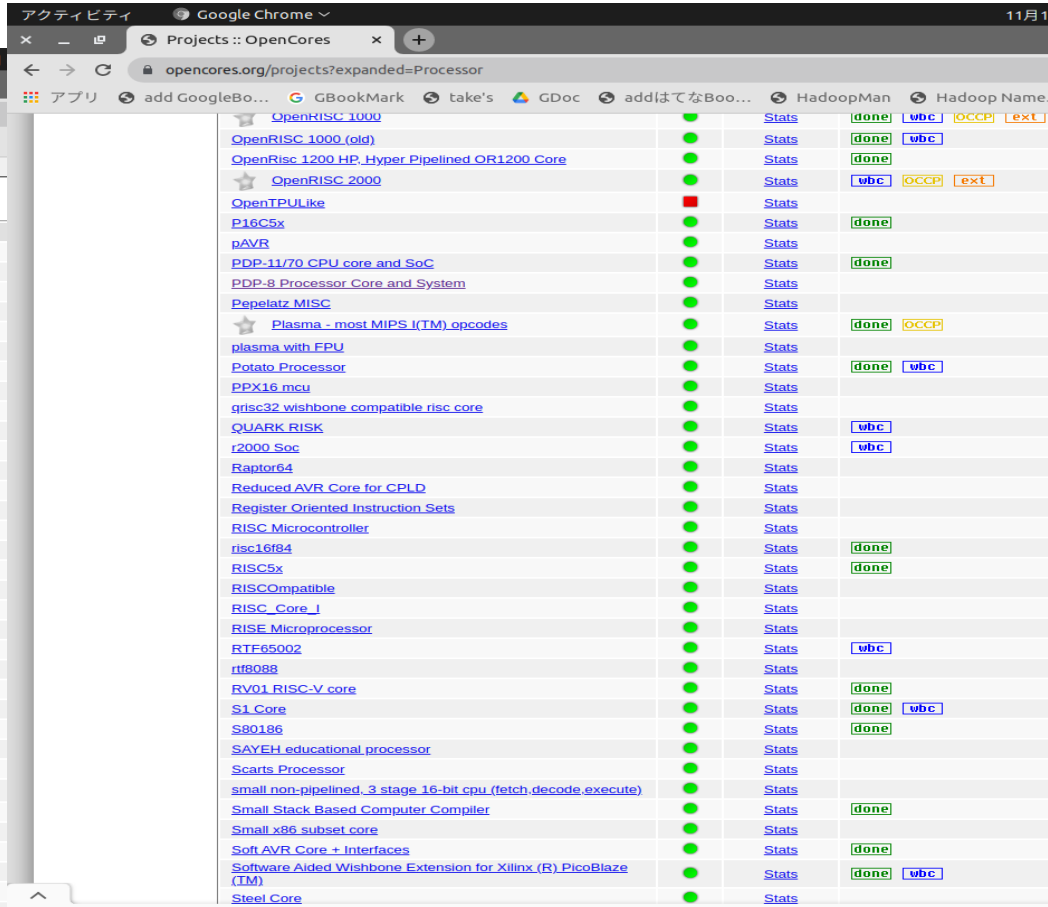
オープンソースなソフトコア

- Opencores

https://opencores.org/

オープンソース・プロジェクトのコア
有名コアのRTL記述 多数アリ ※ライセンスに注意





以上