# 特集
# RISC-V

ETラボ
SHコンサルティング
日本アイ・ビー・エム シニア・リサーチャー 宗藤誠治
FPGA開発日記
須崎有康

産業技術総合研究所

TRY IT NOW

## ハードだってオープンソース、パラダイムシフトを牽引する新世代プロセッサー「RISC-V」

ETラボ　横田英史

**x86とArmが席巻していたマイクロプロセッサーの世界が変わるかもしれない。命令セットアーキテクチャー(ISA)をオープンソースにしたRISC-Vが、ハードウェアのLinuxとして着実に地歩を築き始めたからだ。RISC-VのISAは、無償で公開され、ライセンス料とLSIごとのロイヤリティーを支払うことなく、誰でも自由に改良・再配布が可能である。RISC-V準拠CPUコアや他のIP、開発環境とともに、オープンソースを集めたWebサイトGitHubで公開されている。アプリケーションに最適な構造をもつユーザー所望のマイクロプロセッサーが、手軽に手に入る世界がすぐそこまで来ている。**

マイクロプロセッサーが今、変革期を迎えている。大きく変わろうとしているのは、マイクロプロセッサーの「命令セットアーキテクチャー(ISA:Instruction Set Architecture)」である。性能や消費電力、コストに大きな影響をもたらし、マイクロプロセッサーの性格を特徴づけるISAに、「RISC-V(リスクファイブと発音)」と呼ぶ新たな風が吹き始めたのだ。

マイクロプロセッサーのISAは、米Intel社のx86アーキテクチャー、英Arm社のArmアーキテクチャーが市場を席巻し、この10年以上にわたって「凪」の状態が続いていた。前者はパソコンやサーバー、後者はスマホをはじめとするモバイル機器やコンピューター周辺装置、制御機器といった分野で80%を超えるシェアを占め、他のISAにつけ入るスキを与えなかった。この構図が、ISAのパラダイムを根底から変えるRISC-Vの登場によって大きく変化しようとしている。

本特集は5部構成で、RISC-Vについて解説する。第1部ではRISC-Vを巡る状況を概観し、第2部で命令セットの仕様や動作モードなどの技術的な特徴を明らかにする。第3部以降は実践編である。第3部ではCPUコアや評価ボード選択のポイント、第4部ではオープンソースの開発環境を活用したFPGAへの実装法、第5部ではセキュアな実行環境TEE(Trusted Execution Environment)について開発者自らが論じる。

## Google、IBM、NVIDIA、Samsung、WDなどが標準化団体に参加

RISC-Vが脚光を浴び始めたキッカケは、RISC-Vの標準化や啓蒙活動を手がけるRISC-V Foundation（RISC-V基金）の設立である。支援環境がこれで整った。

設立時には、米Google社や米IBM社、米NIVDIA社、米Western Digital（WD）社といった大手企業が名を連ね、一躍注目をされるようになった。その後も米Hewlett Packard Enterprise社、米Seagate社、米Tesla社、韓国Samsung Electronics社、台湾TSMC、中国Alibaba Groupなどそうそうたる企業が加わった。2018年12月末時点でRISC-V Foundationに参加する企業・団体・教育機関は200ほどに達している（図1）。

特にインパクトが大きいのがWestern Digital社とNVIDIA社の動きである。Western Digital社は、2019年あるいは2020年以降に発売するHDDとSSDのコントローラーのCPUコアをRISC-Vに全面移行することを2017年11月に明らかにした。1年間の出荷台数はCPUコアにして10億個に達する。人工知能向けGPUの最大手であるNVIDIA社も積極的だ。GPUを制御するコントローラー・ユニットとしてRISC-Vを採用すると発表したほか、2018年12月には同社の深層学習アクセラレーター（NVIDIA Deep Learning Accelerator：NVDLA）および推論エンジンとRISC-Vとを組合わせたプラットフォームを公表した。

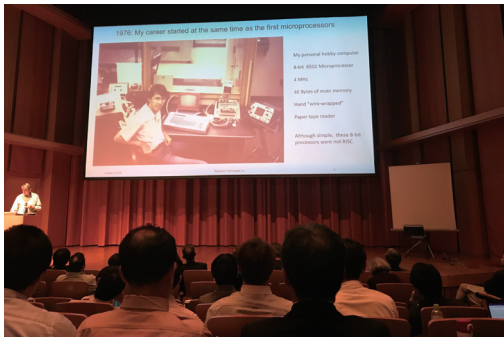


**図2 日本で開催されたワークショップRISC-V Dayの会場風景**
左は2017年12月に東京大学伊藤謝恩ホール、右は2018年10月に慶應義塾大学藤原洋記念ホールで開かれたRISC-V Dayの様子。企業、大学などから前者は350人、後者は250人が出席した。

実際、RISC-Vへの関心はこのところ急速に高まっている。RISC-V Foundationが2015年から半年に1回の頻度で開いているRISC-Vワークショップは多くの参加者を集めている。2018年12月に米国カリフォルニア州サンタクララでRISC-V Summitと題して開催し、約1000人の参加者を集めた。ほぼ1年前の第7回ワークショップの参加者が500人ほどだったので倍増したことになる。

日本の動きも急だ。日本版RISC-Vワークショップ「RISC-V Day」は2017年12月と2018年10月の2回開かれ、それぞれ350人と250人の参加者を集めた（図2）。特に第1回は、「Western Digital社がRISC-Vを全面採用」というニュースが飛び込んだこともあって、会場の東京大学伊藤謝恩ホールは満員で立ち見が出るほどだった。


**図1 RISC-V Foundationのメンバー**
RISC-Vの仕様策定や標準化、啓蒙活動などを推進する、2015年設立の非営利団体。2018年末の時点で約200の企業、団体、教育機関が加盟する。出典：RISC-V SummitにおけるRISC-V FoundationのRick O'Connor氏の講演「Welcome & RISC-V ISA & Foundation Overview」（https://content.riscv.org/wp-content/uploads/2018/12/Welcome-RISC-V-ISA-Foundation-Overview-Rick-OConnor.pdf）

## オープンソースでハードウエアのLinuxを目指す

パラダイムシフトをもたらすRISC-Vの特徴は大きく2つある。1つはオープンソースという点。もう1つはDomain Specific Architecture（ドメイン固有アーキテクチャー）と呼ぶ設計思想を取り入れている点だ。

RISC-Vは、米カリフォルニア大学バークレー校（UCB）が研究目的で開発したISAである。校内で使うことを想定していたが、部外者が利用するようになったことから仕様をオープンソースとして公開し、商業製品にも利用しやすいBSDライセンス条件のもとで誰でも自由に使えるISAとした。つまり無償で公開され、使用料（ライセンス料）とLSIごとのロイヤリティーを支払うことなく、誰でも自由に改良・再配布が可能である。マイクロプロセッサーを独自に開発する場合に気になる特許侵害の懸念も払拭できる。

こうすることでRISC-VはISAを使う敷居を下げ、企業や組織の枠を越えたコミュニティーの衆知を集めて、低コストでの技術の開発と改善を促す。オープンソースのOSとして確かな地歩を築いたLinuxと同様の考え方である。端的に言えば、RISC-Vは「ハードウェアのLinux」を狙っているのだ。RISC-Vは実際、コミュニティーによって改良が続けられた。現在は、RISC-Vを採用したCPUコアやSoCといった他のオープンハードウェアとともに、オープンソースの情報を集めたWeb

サイトGitHubで公開されている。このあたりは第3部以降で紹介する。

ちなみに、ここにきて様々な業界団体のRISC-Vへのシフトが加速し始めた。例えばRISC-V Foundationは2018年12月に、Linuxの普及を促進する非営利団体Linux Foundationとの協業を発表した。両Foundationは、RISC-Vプロセッサーを使ったシステムのLinuxへの対応だけではなく、組込みOS「Zephyr」の開発・普及でも協力体制を敷く。さらに2019年1月にはRaspberry Pi Foundationが、RISC-V Foundationへの参加を明らかにした。

## Domain Specific Architectureの設計思想を取り込む

台頭の背景にある第2の要因は、半導体の技術的問題に対応する潜在能力を秘めていることだ。マイクロプロセッサーの性能向上が壁にぶつかっているのは図3と図4を見るとよく分かるが、RISC-Vにはこの問題を想定したDomain Specific Architectureの仕組みが組込まれている。特定分野に的を絞った機能や仕組みをプロセッサーに組込むことで、対象となる分野について高速処理を達成する。

マイクロプロセッサーの性能を向上させる手っ取り早い方法は、半導体技術の微細化に応じて動作周波数を高めることである。ところが消費電力が大きくなりすぎるために、動作周波数を高められなくなってきた。図3から、2005年ころを境に動作周波数の向上に急ブレーキがかかっているのが分かる。2005年からしばらくは、1つのLSIに複数のプロセッサコアを内蔵し、並列処理を行うことで性能を高めてきた。しかし、それも限界に近づきつつある。

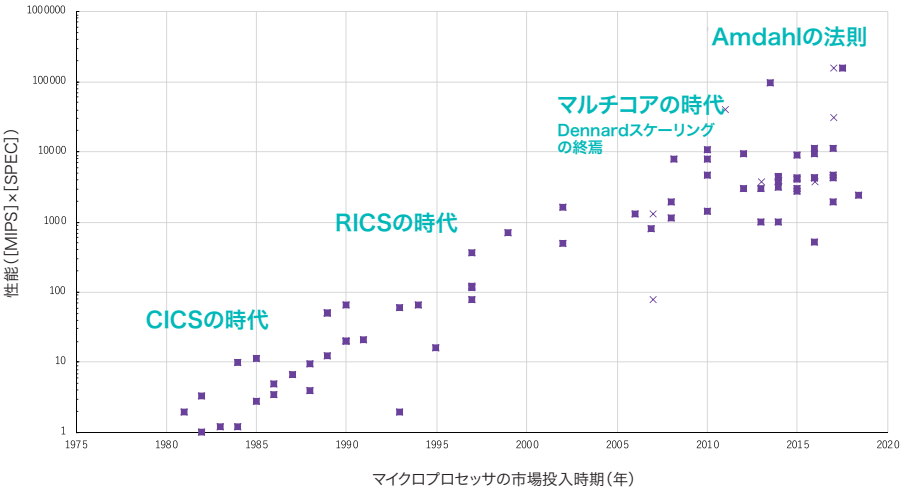
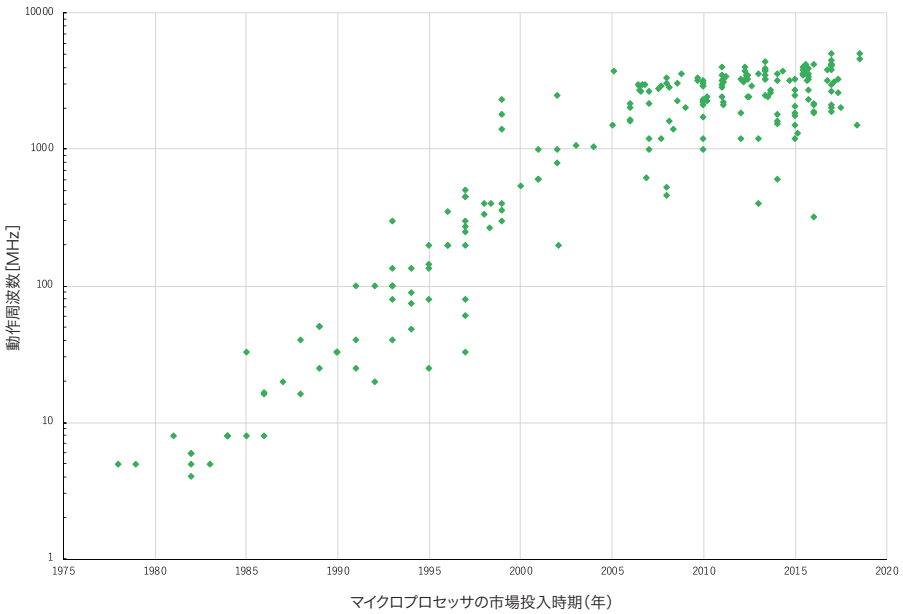



**図4 マイクロプロセッサーの処理性能の推移**
CISCからRISC、マルチコアといった手法で性能を高めてきたが、ここにきて向上のペースは鈍っている。こうした状況をドメイン固有のアーキテクチャー（Domain Specific Architecture）を取ることで補う。Patterson氏によるとCISC時代は年率22%、RISC時代は同52%、マルチコアの時代は同12%で性能は向上していた。しかしAmdahlの法則によって、コア数を増やしても性能が高まらない時代に突入している。現在は年率3%程度しか性能は高まっていないという。
RISC-V SummitにおけるDavid Paterson氏の基調講演「A New Golden Age for Computer Architecture: History, Challenges and Opportunities」（https://content.riscv.org/wp-content/uploads/2018/12/A-New-Golden-Age-for-Computer-Architecture-History-Challenges-and-Opportunities-David-Patterson-.pdf）を参考に作成。

**図3 マイクロプロセッサーの動作周波数の推移**
2005年を境に動作周波数はほとんど横ばい状態である。




こうした状況に対応するのが、マイクロプロセッサーを特定の処理向けに最適化するDomain Specific Architectureである（図4）。最近になって米Amazon.com社や米Microsoft社、Google社といった情報技術（IT）の巨大企業が、AIやクラウドコンピューティングに向けた専用チップの開発に動いているのも、この流れでとらえることができる。

RISC-VのISAには、ドメイン固有の処理に対応した命令を組込むための余地が開発当初から残されている。市場規模にもよるが、自動車や家電製品、制御装置、製造装置といったドメインに特化したRISC-Vチップが登場する可能性もある。UCBの元教授で、RISC-V開発者の一人であるDavid Patterson氏（現在はGoogle Research）はRISC-Vの設計思想について、著書「RISC-V原典」（日経BP社刊）でこう述べている。

ムーアの法則が通用しなくなったときにコスト・パフォーマンスを大きく向上させる唯一の道は、特定のドメイン向け命令を追加することである。例えばディープラーニング、拡張現実、組み合せ最適化、グラフィックスなどのドメインが考えられる。つまり今日のISAにとって、オペコードの拡張の余地を予め確保しておくことが重要になっている。

# RISC-VのISAと普及活動の状況

SHコンサルティング 代表　河崎 俊平

命令セット仕様(Instruction Set Architecture:ISA)は、特許で独占的使用権を確保する。
ISAをオープンソースとし使用権を制限されないISAとしてRISC-Vが登場した。広い汎用性を持たせ全応用に適用できる。
本記事では、RISC-VのISAの特徴とプログラム環境整備状況、IoTやAIのエッジデバイスセキュリティー保護への取り組みを紹介する。

RISC-Vでは、32ビット以上の全応用領域をカバーするため、モジュラーアーキテクチャー方式を採用している(図1)。基本ワード長、命令機能、特権モードなどは応用を考慮し選択実装することで各応用を経済的に実装できる。

RISC-V ISAの特権モードでは、「マシンモード(最高特権)」が必須で「スーパーバイザーモード」「ユーザーモード(最低特権)」はオプションだ。仮想マシン上でリッチOSを実行する時は、3つの特権モードすべてを実装する。マイコンは概ね「マシンモード」のみで実装されている。

RISC-Vの基本命令長は32ビットだが、命令フィールド内の2ビットで命令長が決まり、16ビット圧縮命令(C)もオプションとしてあり、命令空間の4分の1が圧縮命令に割り当てられる。命令密度はThumb命令すなわちArm Cortex M0と理論的には等価である。CSiBEベンチマークは、gccコンパイラ設計者がコード最適化に使うベンチマークであり、893種のベンチマークがアーカイブされている。Cortex M0とRV64GCでCSiBEを2016年頃の未熟なGCCでコンパイルした。平均コードサイズは、RV64GCはM0比15%増、RV32GCは同11%増だった。

M0、RISC-V ISAでは、コード効率ベンチマークで得意不得意がはっきりする。M0が得意とするのは、Linuxカーネルタスク初期化(対RV64GCでサイズが52%)、組込みTCPIPプロトコル(同43%)などである。RV64GCが得意とするのは、MPEG映像データ変換(同20%)、線形代数演算が同38%などである。

## 64ビットのみのRISC-VGCCとLinuxサポート

RISC-Vチームは、RV64のbinutils、gcc、glibc、Linuxをアップストリームし ABIを安定させる作業を進めている。

Linux対応RV32ハードとLinux試作版が2016年ごろ存在したが今は姿を消した。DDR実装量が増加して4GBの物理メモリーは珍しくない。ラズベリーパイですら1GBのDDRを標準実装する。Linux対応Arm32は価格が暴落した。ラズベリーパイゼロボードは5米ドルで売られている。RedHat/Fedoraはサーバークラスの RV64ハードウェアにしか興味がない。Debianサポートに必要な、binutils、gcc,llvm、glibc、Linux、ミドルウェア、プログラミング言語などの上流サポートには年間億単位の費用が必要なため、RV32までケアできない。

## RISC-VへのDebianによる関与の歴史

Debianでは、安定版(stable)とし、テスト版(testing)、不安定版(unstable)の3世代のディストロを並行に公式リリースする。2019年3月時点の安定版DebianはGNU/Linux 9コードネーム「stretch」である。2019年からGNU/Linux 10コードネーム「buster」に移行する。

ABI問題、法律問題などでRISC-Vソフト移植はしばらくゆっくりだった。それらの問題が解決するとRISC-Vツールチェーンのアップストリーム化は驚異的速さで進ん

図1 RISC-Vのモジュラーアーキテクチャー使用例(Linuxが走るRV64GCの場合)

Linuxを実行できるマシンとするには、
①レジスタ長(XLEN)=64ビット。
②命令群としてG(汎用命令群)=
I(整数)+M(乗算)+A(アトミック)
+F(単精度浮動小数点)
+D(倍精度浮動小数点)
+C(圧縮命令)を選択。
④特権モード構成は、
3レベル(3)を選択。
M(マシンレベル)
S(スーパバイザレベル)
U(ユーザレベル)
をサポートする。

| MXL | XLEN |
|---|---|
| 1 | 32 |
| 2 | 64 |
| 3 | 128 |

①レジスタ長の指定

Table 3.1: Encoding of MXL field in misa

| Level | Encoding | Name | Abbreviation |
|---|---|---|---|
| 0 | 00 | User/Application | U |
| 1 | 01 | Supervisor | S |
| 2 | 10 | *Reserved* | |
| 3 | 11 | Machine | M |

特権モード定義

Table 1.1: RISC-V privilege levels.

| Number of levels | Supported Modes | Intended Usage |
|---|---|---|
| 1 | M | Simple embedded systems |
| 2 | M, U | Secure embedded systems |
| 3 | M, S, U | Systems running Unix-like operating systems |

④特権モード構成の指定

Table 1.2: Supported combinations of privilege modes.

| Subset 汎用命令群G | Name |
|---|---|
| Standard General-Purpose ISA | |
| Integer | I |
| Integer Multiplication and Division | M |
| Atomics | A |
| Single-Precision Floating-Point | F |
| Double-Precision Floating-Point | D |
| General | G = IMAFD |
| Standard User-Level Extensions | |
| Quad-Precision Floating-Point | Q |
| Decimal Floating-Point | L |
| 16-bit Compressed Instructions | C |
| Bit Manipulation 圧縮命令C | B |
| Dynamic Languages | J |
| Transactional Memory | T |
| Packed-SIMD Extensions | P |
| Vector Extensions | V |
| User-Level Interrupts | N |
| Non-Standard User-Level Extensions | |
| Non-standard extension "abc" | Xabc |
| Standard Supervisor-Level ISA | |
| Supervisor extension "def" | Sdef |
| Non-Standard Supervisor-Level Extensions | |
| Supervisor extension "ghi" | SXghi |

②命令群の指定

Table 22.1: Standard ISA subset names.

だ。アセンブラやリンカ(binutils)は2017年3月に、コンパイラー(gcc)は2017年5月に、ライブラリー(glibc)は2018年2月にリリースされた。エミュレーター(qemu)も2018年4月にqemu 2.12の一部としてリリースされ、デバッガ(GDB Linux)、高性能コンパイラー(LLVM/Clang)、ジャバJIT(Java/OpenJDK JIT)、Rust言語、Go言語、フリーパスカル、V8(GoogleのJavascriptエンジン)はRISC-Vに移植中である。

## RISC-V Debian自動ビルドの状況

RISC-Vを含むi386以外の全アーキテクチャーのパッケージ・コンパイル状況はDebianの自動ビルダーネットで集中管理されている。

RISC-Vは:①上流ツールチェーン・サポート体制が存在する。②Debianパッケージ管理システムの不安定新アーキテクチャー(dpkg/unstable)サポート体制が存在する。③自動ビルダ・ネットワークが稼働し新パッケージがビルドできる。このためRISC-VのDebian格付けは「Debian第2クラス・アーキテクチャー」別名「Debian移植アーキテクチャー群(debian-ports)」である。

RISC-Vを「Debian正式アーキテクチャー(Regular Debian Architecture)」に格上げするには、以下の追加条件を満たす必要がある。④アーカイブ中の95%以上のパッケージが正しくビルドされる。⑤Debianアーカイブ中に「未リリース」パッケージがない。⑥Debianパッケージ管理システム安定版でアーキテクチャーサポートが利用できる。⑦Debianシステム管理システムにつながる物理ハード(ボード、ボックスなど)上で全ビルドが動作する。⑧移植作業インフラ(Porterbox)が利用可能である。

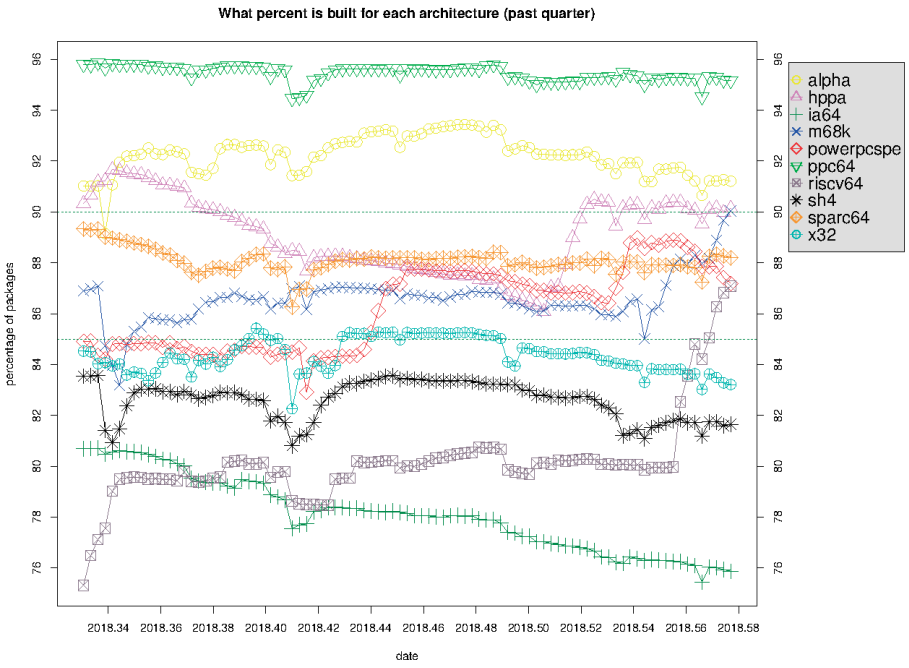



図2　2018年前半期における各アーキテクチャーのビルド成功率

図2は、2018年前半期時点での各アーキテクチャーのビルド成功率(Y軸)を示す。RISC-Vは、2018年半ばに成功率を80%から88%に急増させた。PowerPC64は、95%を維持している。SH4は、アーキテクチャー責任者(メンテナ)が無給与なのでビットロット状態が続き81%~84%にまで下落した。Trust Moduleに使用するためSH用gccツールを開発したが、Debianが使用するアップストリーム化の予定はまだない。

## 「リリースアーキテクチャ」への道

Debianは、安定版リリースが出る度に「リリースアーキテクチャー」を認定する。RISC-Vが認定されるためには、次の条件を満たす必要がある。(ⅰ)Debianインストーラ(Debian-Installer)のサポート。(ⅱ)buildに十分な冗長性を持つ。(ⅲ)Debian自動ビルダーネットが管理するハードがラックマウント可能でゾーン外リモート管理機能を持ち、スペアハードウェア準備がある)。(ⅳ)アーキテクチャー固有の問題を処理する最低3人の開発者をアサインする。

現在進行中のDebian10、通称「Buster」では「遷移フリーズ」「ソフトフリーズ」「完全フリーズ」が、2019年第1四半期に始まるのでRISC-Vは間に合わない。「リリースアーキテクチャー」となれるのは、少なくともDebian11以降である。RISC-VのようなオープンISAとオープンソースのCPU IPがDebian「リリースアーキテクチャー」となれば、RISC-Vの目指す頂点=サーバーに一挙に近づく。

### オープンSoCを支援する「チップ連合」設立

Linux Foundationは下部組織として「チップ連合」(CHIPS Alliance)を2019年3月に発足した。企業と個人が協力しオープンCPUとシステムオンチップ(SoC)の設計資源を作り出す。対象分野は①モバイル、②コンピューティング、③家電製品、④IoT、である。Google社、Western Digital社、Esperanto社、SiFive社が後援する。

AI、機械学習、インフラ向け計算基盤をRISC-Vが担うにはオープンSoCの解が必要である。RISC-Vに加え、まちまちなライセンス条件や開発言語で散在するオープンハードをSoC基盤ブロックにまとめる努力も重要だ。

OpenRISC、レトロCPU(例:Z80)、グラフィックスIP、セキュリティーIP、アナログIPを整備する動きもある。オープンソースIP基盤ブロックが必要である。Google社は、RISC-V向けに、高ストレスランダム検証命令ストリーム生成ツールを提供することをコミットした。

10年の計を持つRISC-Vには安定した政治地盤も重要だ。基盤を欧州に持つLinux Foundationに合流することで、グローバルな組織基盤を確立し、米中半導体戦争などの影響を受けづらくする。RISC-V Foundationは米国基盤でISA規格管理とイベント運営を担う。複眼的な組織構成になりつつある。

### 日本でのRISC-Vの取り組み

筆者は2014年夏からRISC-Vのイベントに参加してきた。RISC-V Foundationは、モーター、ロボット、民生応用の聖地である日本は、RV64の中位製品を開発することを期待した。2016年頃欧米アジアと日本の間にRISC-Vに関する大きな情報格差があることに危機感を持ち、SHコンサルティングとしてRISC-V Foundationに加入した。その後、2017年にはテカナリエ社、2018年にはNSITEXE(デンソーのグループ企業)、ペジーコンピュータ、2019年にはソニーセミコンダクタソリューションズが加入した。

2017年12月にRISC-V Foundationから要請があり「RISC-Vの1日」と呼ぶ年1回の国内の催しを開催した。第1回の開催後に関係者が集まり反省会を行なった。そこでは、(a)日本発のRISC-Vの発表、(b)日本のRISC-Vチップ開発、(c)日本語RISC-V書籍、(d)日本RISC-V法人が必要という意見が出た。これを受けて、2018年に活動を加速した。その結果(a)(b)(c)は実現できた。2019年は、東京国分寺に新設される日立馬場記念ホールを使わせていただき9月30日に開催する。

RISC-Vは、当初バイエンディアンとして誕生した。後に、ビッグエンディアンを仕様から削除した。4回に渡りビッグエンディアン追加要求があったが、ツールがネックで実現しなかった。しかし、通信インフラ、ベースバンド、工場インフラ、電力インフラ、航空、軍事、宇宙応用などはビッグエンディアンが主流だ。Debian「リリースアーキテクチャー」のうち、arm64、arm(32)、MIPS、ppc64elはバイエンディアンでLinuxもビッグリトルを両方完全にサポートしている。Armのビッグエンディアンは、iPhoneのベースバンドチップに使われている。エンディアン間のソフト移植はエラーが生じやすいため、ビッグエンディアンRISC-Vのニーズはある。近くRISC-Vビッグエンディアン・コンパイラーの調査移植を行う。

## オープンな仕組みを活用してエッジデバイスのセキュリティーを守る

つながるデバイスのセキュリティー技術をソフトとハードのオープンソース・パッケージとして展開する活動を、NEDOの「高効率・高速処理を可能とするAIチップ・次世代コンピューティングの技術開発」プロジェクトで進めている。将来のIoTやAIエッジデバイス用SoCでの利用を目指したものだ。日立製作所、慶應義塾大学、産業技術総合研究所、SHコンサルティング、セコム、電気通信大学、東京大学が参加している。このプロジェクトでは、アプリAPIから応用プログラマがシステムセキュリティーを担保するトラスト実行環境(Trusted Execution Environment:TEE)と物理耐タンパ性を持つTrust Module、産業用途、セキュリティーインフラなどの応用技術をテーマとしている。

### 2段構えでエッジデバイスを守る

TEE(Trusted Execution Environment=トラスト実行環境)は、メインCPU群で動作するリッチOS(例:Linux)の追加機能である。TEE API群を介しメモリー隔離ハード機構(TEE)を使う。TEEのAPIは抽象化さ

れているのでハードが変わってもアプリのセキュリティースキームは移行できる。

よく、TEE以外になぜTrust Moduleが必要なのか、と聞かれる。TEEは、リッチOSとTEEが正しく立ち上がった時だけ機能する。TEEが搭載されてないチップ単体に固有鍵を注入することもある。ボードに組み込まれた後、IoTを個人化(Personalization)する。LinuxやTEEが動作しないIoTデバイスを退役させることもTrust Moduleの仕事だ。また、デバッグのためにICEを繋げることもあるが、これも固有鍵を使い暗号化された通信を行う。このようにTEEが起動される前後に多様なライフサイクル処理がある。

Trust Moduleは、鍵注入処理、バイオメトリ情報、セキュアブート機能、デバッグ機能、ストレージ暗号化機能などを、モジュール単体で行える独立したコンピューター(マイコン)である。PKIアルゴリズムを使い、統一された暗号ワークフローで鍵管理を安全に行う。Trust Moduleは、海外政府暗号調達基準(FIPS140-2など)の物理セキュリティー規定に対応できる。FIPS140-2暗号境界(TOE境界)を半導体のマクロセル境界を合致させ、境界内のハードとソフトを独立認定する。応用システムを認定せずとも、Trust Module認定で調達基準を充足できる。

図3に、Trust Moduleのアーキテクチャーを示す。⑥のSoCは、3つのサーバーと1つのデスクトップと1つのセーフボックスから構成される。

各要素の説明を簡単にする。

①ハードウェアトラストモジュール(HTM)サーバー:Trust Moduleの不揮発性メモリーにチップ製造工程で書き込む固有鍵を高速に生成し、PBXに安全に送る。固有鍵が正しく書き込まれたかもチェックする。

②認証(CA)サーバー:Trust Moduleが生成したデジタル署名を検証する。

③データサーバー(DS):Trust Module内の資格証明書を使い、ユーザーセキュリティー運用を行う。

④トラスト開発デスクトップキット(TDK):Trust Module用アプレットと、コアプレックス用アプリを開発するためのツールキットをホストするPC。

⑤パーソナリゼーションセーフボックス(PBX):Trust Moduleに鍵と個別化データを注入する。Trust Module外部ピンに電気的につながるテスター、プローバ、ハンドラーとのインタフェースを備える。HTMと認証交信する機能も持つ。使用状況によりタンパ検出機能も持つ。

⑥SoC:リッチOS(Linux + TEE)を実行できるチップ。

⑦コアプレックス:CPUコア群。マルチコアとMMU、L1キャッシュ、L2キャッシュの集合体のこと。

⑧Trust Module:セキュアMCUマクロセル。セキュアOSを実行する。セキュアMCUは、乱数生成、暗号アクセラレーター、耐タンパ機能を持つ。セキュアOSは、自己充足的なコンピュータシステムでユーザープログラムをダウンロード実行できる。セキュアOSは、暗号ライブラリーを機能の一部として持つ。

⑨メイルボックス:コアプレックスからTrust Moduleのメモリー空間をアクセスすることはできない。セキュアMCUは、メイルボックスを介してコアプレックス空間をバスマスターとしてアクセスすることができる。

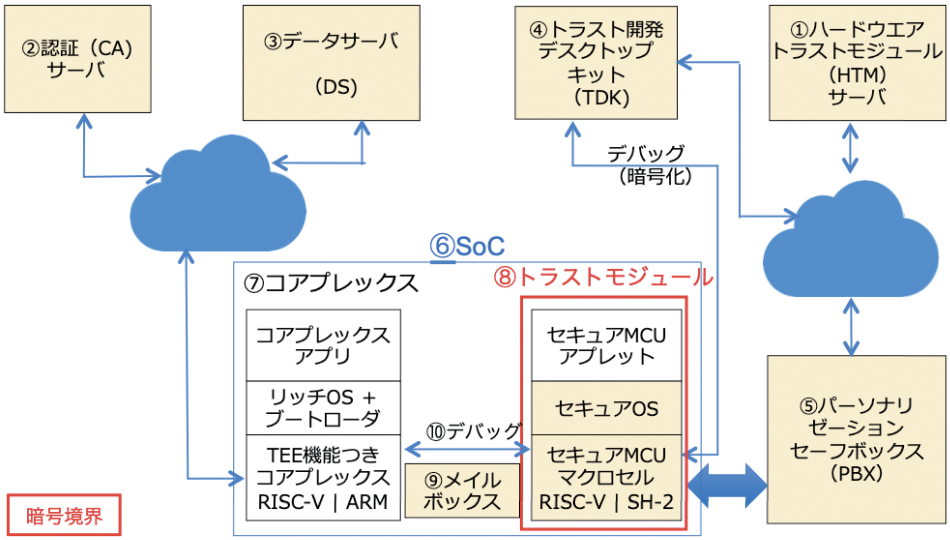



図3 トラストモジュール・アーキテクチャー



掲載した研究の一部は東大VDECの設備とファブを使用させていただいた。

参考文献:Karsten Merker , Porting Debian to the RISC-V architecture , Tales from a long quest , February 2. 2019

# RISC-Vベースのエッジ向けプロセッサー

日本アイ・ビー・エム　シニア・リサーチャー　**宗藤 誠治**

**IoTのエッジ向けプロセッサーでは英Arm社の「Cortexシリーズ」が広く利用されている。
その一方で、近年、オープンな命令セットを採用した「RISC-V」という新しい選択肢が注目されている。
RISC-Vの普及とともに、さまざまなRISC-Vのコアが発表され、選択肢が広がってきた。
ここでは、現在利用可能なRISC-Vの各種実装の紹介と、選択のポイントについて解説する。
特にスイス連邦工科大学チューリッヒ校（ETH Zurich）で開発が進められている「PULP」と、
米SiFive社の「Freedom」についてその概要を紹介する。**

RISC-Vはオープンな命令セット（Instruction Set Architecture：ISA）である点が特徴である。つまり、誰でも自由にRISC-VのISAをサポートしたプロセッサーを開発することができる。開発したプロセッサーはオープンソースとして公開することも、商用製品として利用することもできる。RISC-VのISAはRISC-V Foundation（https://riscv.org/）によって管理されており、基本的には、各社のRISC-Vプロセッサー上で同じバイナリコードを実行することが可能である。コンパイラーやオペレーティング・システムのRISC-V対応も進んでおり、ソフトウェア開発もその他のISAを利用した場合と同等になりつつある。

RISC-VのISAは32ビット、64ビット、128ビットに対応しており、組込みでは32ビットのコアが利用される場合が多い。RISC-VのISAにはいくつかのグループがあり、サポートする命令によってコアのサイズが異なる。そのため、RISC-Vのプロセッサーコアでは、対応するISAを「RV32IMC」のような形で表記する。RV32IMCとは、32ビットで整数（Integer）、乗算器（Multiplier）、縮小命令セット（Compressed）に対応したプロセッサーコアであることを示す。

組込みの場合、利用できるメモリーサイズが限られていることから、バイナリコードが小さいことが重要になる。縮小命令セットを用いた場合、Arm社のCortexシリーズとほぼ同等のコードサイズとなる。さらに、組込み用に命令を絞ったE（Embedded）命令セットも用意されている。

## 市販のSoCを利用したソフトウェア開発

市販されているRISC-Vを採用したプロセッサーはまだ多くはない。それでも、表1に示すように、いくつかのプロセッサーと評

表1　RISC-VのSoCを搭載した評価ボード一覧

| 評価ボード | HiFive1 | HiFive1 rev B | KD233 | VEGA |
|---|---|---|---|---|
| 価格 | $59 | $59 | $49.99 | 無料 |
| 開発元 | SiFive(米) | SiFive(米) | Kendryte(中) | NXP(蘭) |
| SoC名 | FE310-G000 | FE310-G002 | K210 | RV32M1 |
| コア | SiFive E31 | SiFive E31 | 不明（2 core） | RI5CY/ZERO-RISCY |
| ISA | RV32IMAC | RV32IMAC | RV64GC | RV32IMC |
| パイプライン | 5-stage | 5-stage | 不明 | 4-stage/2-stage |
| 命令メモリー(キャッシュ) | 16KB | 16KB | 32KBx2 | 4KB |
| データメモリー | 16KB | 16KB | 32KBx2 | |
| その他SRAM | | | 8MB | 256KB/128KB |
| アクセラレータ | | | KPU, APU, FFT, AES, SHA256 | AES, DES, SHA256, RSA, ECC |
| 外部インターフェース | GPIO, UART, PWM, SPI | GPIO, UART, PWM, SPI, I2C | GPIO, UART, SPI, I2C, I2S, PWM | GPIO, UART, I2C, SPI |
| ASIC | TSMC 180nm | | TSMC 28nm | |
| 動作周波数(MAX) | 320MHz | | 400MHz | 72MHz |
| SW開発環境 | SDK, IDE提供 | SDK, IDE提供 | SDK提供 | Toolchain提供 |
| OS | FreeRTOS, Zephyr | FreeRTOS, Zephyr | FreeRTOS | FreeRTOS, Zephyr |
| URL | https://www.sifive.com/boards/hifive1 | https://www.crowdsupply.com/sifive/hifive1-rev-b | https://kendryte.com/ | https://open-isa.org/ |

価ボードが入手可能になってきた。こうした評価ボードを利用することで、RISC-V向けのソフトウェアの開発を実機上で進めることができる。SiFive社の「HiHive1」は、日本国内でも入手可能なArduino互換ボードであり、RISC-V上で何かソフトウェアを動作させるにはお手頃な製品である。各種の開発ホスト環境向けのクロスコンパイラーを含むSDKもリリースされており、Arduino IDEを使い、自由にプログラムの開発&実行ができる。

## 独自SoCの開発

RISC-Vを採用する動機の一つが、独自のSoC開発であろう。RISC-Vには特定のアプリケーションに特化して最適化を進めるドメイン固有アーキテクチャー(Domain Specific Architecture)を実現するための、最も先進的な仕組みが備わっている。コアに関しても、表2に示すように様々なコアが様々な組織(大学、企業)からオープンソースとして公開されている。また、商用のコアも発表されている。

2015年にETH ZurichがPULPinoをGithub上でリリースした。その後、2016年にカリフォルニア大学バークレー校のRISC-V開発チームが立ち上げたSiFive社により開発されたFreedomもGitHub上で公開された。2018年末には米Western Digital社がRISC-Vのコアを公開している。今後も各社で開発したRISC-Vのコアを一般に公開する動きが続くと思われる。選択肢が増えることは好ましいが、利用に際しては自分たちに適したコアがどれなのかを見極める必要がある。

| | |
|---|---|
| | GAPUINO |
| | 100,00€ |
| | GreenWaves Technologies(仏) |
| | GAP8 |
| | RI5CY (8 core) |
| | RV32IMC |
| | 4-stage |
| | 64KB(L1) |
| | |
| | 512KB(L2) |
| | CNN |
| | UART, SPI, I2C, I2S, PWM, GPIO |
| | TSMC 55LP |
| | 250MHz |
| | SDK提供 |
| | FreeRTOS |
| | https://greenwaves-technologies.com/product/gapuino/ |

## RISC-V実装の選択のポイント

RISC-Vのプロセッサーの開発言語としては、SystemVerilogとChiselの二つが広く用いられている。一般的なハードウェア開発では、SystemVerilogの利用が一般的であり、SystemVerilogで記述されたRISC-V実装は多くのエンジニアに受け入れられやすい。またAXI/APBのようなバスを採用している場合が多く、既存のIPを組込みやすい。

一方、Chiselはカリフォルニア大学バークレー校で開発された高位のハードウェア記述言語であり、Scala言語をベースに設計されたドメイン特化言語(DSL)である。同校で開発されているRISC-V実装の「Rocket-core」はChiselで記述されており、先に述べたSiFive社のFreedomもChiselで記述されている。この場合、内部バスはTileLinkとよばれる独自のものである。Verilogなどで書かれたIPを組み込むことは可能であるが、Chiselの開発環境での検証には制約がある。現状ではChiselに慣れた開発者が少ないことが課題と言える。今後ドメイン固有アーキテクチャーを実現するために、命令を追加したり、コ・プロセッサーを組み合わせたりする作業には、カリフォルニア大学バークレー校が開発した新しい言語と開発環境の利用は検討の価値がある。

こうしたコアの選択のポイントとしては、以下の項目について注意する必要がある。

- メモリー構成
- セキュリティー機能
- デバッグ機能(JTAG)
- ソフトウェア・サポート

### メモリー構成

一般に、命令実行用とデータ用の二つの内部メモリー(SRAM)が必要となる。これらがキャッシュとして実装される場合は、プロセッサーチップの外部に、より大きなメモリーを接続することとなる。アプリケーションは外部に接続されたフラッシュメモリーなどに保存される。

メモリーマップはそれぞれの実装で異なるため、ソフトウェアで複数のSoCをサポートする場合は注意が必要であるが、これはArmコアを用いる場合でも同様である。

### セキュリティー機能

IoTのエッジデバイスではセキュリティーの確保が重要になる。RISC-Vのコア自体のセキュリティー機能としては特権レベルとメモリー保護機能(Physical Memory Protection:PMP)がある。現時点では、暗号処理向けの命令セットはまだ策定中であり、暗号処理を高速化、低消費電力化するためには、独自命令の追加、暗号アクセラレーターの追加などが必要である。

表2 エッジ向けのRISC-Vコア一覧

| SoC名 | PULPino | PULPissimo | | | | Freedom E300 | | | | | Briey | PicoSoC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| コア | RI5CY | RI5CY | Zero-Riscy | Micro-Riscy | Z-Scale | E300 | E51 | E31 | E24 | E21/E20 | | PicoRV32 |
| 開発元 | ETH Zurich | ETH Zurich | ETH Zurich | ETH Zurich | UCB | SiFive | SiFive | SiFive | SiFive | | SpinalHDL | Clifford Wolf |
| ISA | RV32IMCX | RV32IMCX | RV32IMC | RV32EC | RV32IM | RV32IMAC | RV64IMAC | RV31IMAFC | RV32IMC | RV32IMAFC/ RV32IMAC | RV32IMC | RV32IMC |
| パイプライン | 4-stage | 4-stage | 2-stage | 2-stage | 3-stage | 5-stage | 5-stage | 5-stage | 5-stage | 3-stage | 5-stage | |
| 内部メモリー | 32KB I-SRAM, 32KB D-SRAM | | | | | I-Cache, D-SRAM | I-Cache, D-SRAM | I-Cache, D-SRAM | | | | 1KB |
| 内部バス | AXI/APB | AXI/APB | | | AHB-Lite | TileLink | TileLink | TileLink | | | | AXI4 |
| Privilege levels | M | M | | | | M,(U) | M.U | M.U | M,U | M | | |
| PMP | N/A | N/A | | | | 16 | Max 16 | Max 16 | Max 16 | | | |
| 設計言語 | SystemVerilog | SystemVerilog | SystemVerilog | SystemVerilog | Verilog | Chisel | Chisel | Chisel | Chisel | Chisel | SpinalHDL | Verilog |
| Dhrystone (DMIPS/MHz) | | | | | 1.35 | | 1.8 | 1.61 | 1.61 | 1.38 | | 0.31 |
| CoreMark (CoreMarks/MHz) | | 3.19 | 2.44 | 0.91 | | | 2.76 | 2.73 | 3.01 | 3.1 | | |
| I/F | I2C, UART, GPIO, SPI | | | | | | | | | | | SPI, UART |
| デバッグ | SPI Slave経由 | | | | | JTAG | JTAG | JTAG | | | | |
| FPGA | ZedBoard | | | | LX9 Microboard | Arty A7 | | | | | DE1-SoC, DE0-Nano | iCE40-HX8K |
| ソフトウェア開発環境 | pulp-builder SDK | pulp-builder SDK | | | | Arduino, Freedom-SDK | | | | | | |
| OS | FreeRTOS | | | | | FreeRTOS, Zephyr | | | | | | |
| ライセンス | Solderpad HW | Solderpad HW | Solderpad HW | Solderpad HW | BSD | BSD | 商用 | 商用 | 商用 | 商用 | MIT | ISC |
| Github | https://github.com/pulp-platform/pulpino | https://github.com/pulp-platform/pulpissimo | https://github.com/pulp-platform/zero-riscy | | https://github.com/ucb-bar/vscale | https://github.com/sifive/freedom | | | | | https://github.com/SpinalHDL/VexRiscv | https://github.com/cliffordwolf/picorv32/tree/master/picosoc |

特権レベルについてはM(Machine)モードとU(User)モードの二つをサポートする場合が多い。これとPMPを組み合わせることで、ファームウェアを保護することができる。RISC-V Foundationでは、一般にTEE(Trusted Execution Environment)と呼ばれるセキュアな実行環境を、こうした機能を用いて実現するための作業が進められている。

### デバッグ機能

プロセッサーのデバッグとして一般的なJTAGのサポートの有無も重要である。OpenOCD経由でデバッグできることが望ましい。

### ソフトウェア開発環境

一般的にはCもしくはC++を用いてソフトウェアの開発を行う。GCCはすでにRISC-Vに対応しており、各ホスト環境に応じたクロスコンパイラーが入手可能である。各種のコアは最低限クロスコンパイラーには対応している。ものによっては、BSPを含むSDKや、IDEが提供される場合もある。OSについてもFreeRTOSやZephyrなどのRISC-V対応が進んでいる。

### ASIC化のサポート

GitHub上で公開されている実装は、主にFPGAでの評価までの情報しか公開されていない。機能評価はFPGAを使った環境で十分だが、最終的にシリコンに落とす際には、各種テクノロジーに依存したライブラリーやメモリー、I/OなどのIPが必要となる。この部分はまだ敷居が高いのが現実である。社内にそうしたサポートがない場合は、商用の支援サービスを利用するのも一つの方法である。近年はクラウド上でASIC化を支援するサービスも始まってきており、将来は簡単にチップが作れる時代がくるかもしれない。

## ETH ZurichのPULPino

ここからは、RISC-Vの実装として代表的なETH ZurichのPULPinoとSiFiveのFreedomについて概要を紹介する。

まずPULPinoであるが、元々ETH ZurichはOpenRISCコアを使った低消費電力プロセッサーの研究を進めており、そのプロセッサーコアをRISC-Vに変更したものが、PULPinoである。そのため公開当初からSoC用コアとして十分な機能を持っていた。またFPGAでの検証環境もサポートしている。設計はSystemVerilogで記述されており、内部バスもAXI/APBであるため、組込みプロセッサーの開発経験があれば簡単に機能の追加や変更が可能である。

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Raven | Icicle | Riscy-SoC | mriscv | ORCA | ArtyS7-RPU-SoC | SHAKTI E-Class | SHAKTI C-Class | SweRV Core | N25/N25F | NX25/NX25F | Piccolo | Mi-V |
| PicoRV32 | Icicle | (Icicle) | mriscvcore | ORCA | RPU | E-Class | C-Class | SweRV EH1 | N25/N25F | NX25/NX25F | Piccolo | Mi-V |
| efabless engineering | Graham Edgecombe | Aleksandar Kostovic | OnchipUIS | VectorBlox | Colin Riley | IIT-Madras | IIT-Madras | Western Digital | Andes | Andes | Bluespec | Microsemi |
| | RV32I | RV64I | RV32I | RV32IM | RV32I | RV32/64-IMAC | RV64IMAFD | RV32IMC | RV32IMAC | RV64IMAC | RV32ACIMU | RV32IMA, RV32IMF |
| | 5-stage | | | 4 or 5-stage | | 3-stage | 6-stage | 9-stage | 5-stage | 5-stage | 3-stage | |
| 4KB | 8KB | | 4KB SRAM | | | | | I-Cache、ECC | I-Cache & D-Cache, 8KB to 64KB | I-Cache & D-Cache, 8KB to 64KB | | 8KB I-Cache, 8KB D-Cache |
| | | | AXI4, APB | | | | | AMBA | AHB or AXI | AHB or AXI | | |
| | | | | | | | | M | M, U | M,U | M,U | |
| | | | | | | | | N/A? | 16 regions | 16 regions | | |
| Verilog | SystemVerilog | Verilog | Verilog | VHDL | VHDL | Bluespec System Verilog | Bluespec System Verilog | SystemVerilog | Verilog | Verilog | Bluespec System Verilog | Verilog |
| 0.31 | | | | 0.98 | | | 1.67 | | 2.85 | 3.22 | | |
| | | | | | | | 2.2 | 4.9 | 3.58 | 3.52 | | |
| GPIO, ADC, DAC, SPI, UART | UART,SPI | UART | ADC, DAC, GPIO, SPI | DE2-115, iCE5LP4K, ZED | | | UART, QSPI, I2C | | | | | |
| | | | | | | | | | | | | |
| | iCE40-HX8K | iCE 40 | NEXYS4 | | Arty S7-50 | Artix7 FPGA | | N/A | | | | M2GL025 |
| | | | | | | | | N/A | AndeSight IDE | AndeSight IDE | | |
| | | | | | | | | | | | | FreeRTOS, Zephyr |
| ISC | ISC | MIT | MIT/BSD | BSD | Apache2.0 | GPL 3 /BSD | BSD | Apache2.0 | 商用 | 商用 | Apache2.0 | |
| https://github.com/efabless/raven-picorv32 | https://github.com/grahamedgecombe/icicle | https://github.com/AleksandarKostovic/Riscy-SoC | https://github.com/onchipuis/mriscv | https://github.com/vectorblox/orca | https://github.com/Domipheus/ArtyS7-RPU-SoC | https://gitlab.com/shaktiproject/cores/e-class | https://gitlab.com/shaktiproject/cores/c-class | https://github.com/westerndigitalcorporation/swerv_eh1 | | | https://github.com/bluespec/Piccolo | https://github.com/RISCV-on-Microsemi-FPGA/M2GL025-Creative-Board |

FPGAの評価環境はXiinx社のZinqを用いる。ARMのコアを経由してRISC-V側にアクセスするので、開発環境の構築に少し手間が掛かる。表1に挙げたVEGAやGAPUINOはコアとしてETH Zurichで開発されたRI5CYを採用している。

## SiFiveのFreedom

一方、Freedomは、SiFive社の組込み用のSoCで、カリフォルニア大学バークレー校が開発しているRocket-coreをベースに、組込みで必要となるペリフェラルを追加し、SoCの形にまとめたものである。ScalaベースのDSL(ドメイン固有言語)であるChiselで記述されており、構成変更が容易である。実際には、ChiselからFIRRTLと呼ばれる独自のRTL記述に変換され、さらにVerilogのRTLコードが生成される。以降は通常の論理合成ツールを用いて、FPGAやASICのネットリストを生成する。Tilelinkと呼ばれる独自の内部バスを採用しており、既存IPの利用には制約がある。

ペリフェラルの追加はChiselでの再設計が望ましい。そのため、新規にChisel言語の習得が必要となるが、Rocket-coreへの命令の追加やRoCCと呼ばれるコ・プロセッサー機能などがあり、ドメインに特化した最適化に適した開発環境である。この他にもSiFive社は表2に示すような性質の異なる商用コアもいくつかリリースしているので、製品で使用する場合はそうしたコアの採用も可能である。表1に挙げたArduino互換のボードも入手可能である。

また、GitHubで公開されているChiselのソースコードを元に実際にFPGAで動作させるまでの作業も体験できる。ソフトウェア開発環境もリリースされており、最も簡単にRISC-Vの世界が体験できる環境を提供している。FPGA版の実装のソフトウェアのアップロードはJTAG、OpenOCD経由で行うため、JTAGアダプターが別途必要となる。

## まとめ

以上、エッジ向けのRISC-Vコア、SoCを簡単に紹介したが、RISC-Vのエコシステムはこの数年で急速に広がってきており、今後一般に入手可能なプロセッサーやボードも増えてくると思われる。そうした既存のチップを使ってエッジ機器を作るだけでなく、SoCレベルで自由に最適化を試みることができる点が、RISC-Vと従来ISAとの大きな違いである。組込みの場合安価なFPGAボードを使い、手軽にRISC-Vの評価を始めることができるので、ぜひオープンなISAのメリットを体験していただきたい。

# RISC-Vを触ってみる。無料で始められるRISC-V開発環境の構築とFPGA実装

FPGA開発日記

「RISC-V」のISAはオープンソースであり、様々な企業や団体がRISC-Vに対応したオープンソースCPUコアを公開している。
RISC-Vを触ってみたい技術者はネット上から情報を入手でき、無料のRISC-Vコアを使用してFPGAに実装したり、
ASICを起こすことができる。
ここでは、RISC-V対応のオープンソースCPUコアを使うための様々な開発環境の紹介と、
その構築方法についてイントロダクションを行う。

2019年現在、ネット上にはRISC-Vに関する多様なリソースが公開されている。RISC-V ISAの仕様書、オープンソースのCPUコア、コンパイラーからLinux、無料の開発環境まで様々だ。技術者は自分の興味のある部分からRISC-Vを触り始めることができる。

RISC-Vをどのように使っていきたいのか。その活用法は人によって千差万別である。以下の項目に注目して、はじめの一歩を踏み出すための手順について解説していきたい。

- とりあえず手元のCプログラムをRISC-V ISAでコンパイルして、命令セット・シミュレーターで動かしてみたい。
- オープンソースのRISC-V CPUコアを使ってRTLシミュレーションを行い、FPGAで動かしてみたい。

筆者がRISC-Vについて調査を始めた2015年当時、RISC-Vのソフトウェア環境とハードウェア環境はほとんど公開されていなかった。しかし、現在はQEMUのようなエミュレーターからLinuxのようなOS、GCC（GNU Compiler Collection）やLLVMなどコンパイラー関連、そして多くの団体から公開されたオープンソースCPUコアまでよりどりみどりの状態になった。ネットを検索すればすぐに多くの情報を入手できるので、今回紹介するもの以外でも興味があれば、ぜひ調べてみてほしい。

## RISC-Vソフトウェア開発環境の構築手順

RISC-Vソフトウェアを開発するためにまず必要なのは、コンパイラーやシミュレーターなどのツール群だ。GitHubに公開されている「riscv-tools」（https://github.com/riscv/riscv-tools）は、これらの環境がすべて入っているリポジトリーである。このリポジトリーをダウンロードすると、とりあえず必要なツール群を一式揃

コード1

```
git clone https://github.com/riscv/riscv-tools.git
cd riscv-tools
export RISCV=/home/msyksphinz/riscv64
MAKEFLAGS="-j8" ./build.sh
...
cd {RISCV}/
export PATH={RISCV}/bin:{PATH}
export LD_LIBRARY_PATH={RISCV}/lib:{LD_LIBRARY_PATH}
```

コード2

```
#include <stdio.h>

int gcd (int a, int b)
{
  int c;
  if (a < b) {
    int tmp; tmp = b; b = a; a = tmp;
  }
  while (b != 0) {
    c = a % b; a = b; b = c;
  }
  return a;
}
int main ()
{
  printf ("Calling GCD ...\n");
  printf ("GCD(411, 27117) = %d\n", gcd(411, 27117));
  return 0;
}
```

コード3

```
riscv64-unknown-elf-gcc gcd.c -o gcd
spike pk gcd
<<set terminal>>
<<reset terminal>>
Calling GCD ...
GCD(411, 27117) = 3
<<reset terminal>>
```

コード4

```
git clone https://github.com/freechipsproject/rocket-chip.git --recurse-submodules
cd rocket-chip
cd emulator   # Verilatorを使う場合
cd vsim       # Synopsys VCSを使う場合
make CONFIG=DefaultConfig  # DefaultConfig構成でビルド(RTL作成)
```

コード5

```
make output/rv64ui-p-add.out
```

えることができる(ただしリポジトリーのサイズは数GBにおよぶので、すべてダウンロードするときはディスク容量に注意して欲しい)。

以下の手順でツールセットをダウンロードしてビルドする。ここで環境変数「${RISCV}(=/home/msyksphinz/riscv64)」を設定しているが、このディレクトリーにツール群がインストールされることになる。(コード1)

では、早速C言語のプログラムを「RISC-V GCC」でコンパイルして、命令セット・シミュレーター「spike」で実行してみる。以下のような最大公約数(gcd)を計算するプログラムをコンパイルしてみよう。(コード2)

「riscv64-unknown-elf-gcc」でコンパイルし、spikeで実行する。spikeはRISC-Vの命令セット・シミュレーターであり、「printf()」呼び出しなどのシステムコールにも対応している。(コード3)

このように、簡単にC言語のプログラムをコンパイルし、RISC-Vシミュレーターでその動作を確認することができる。

## Rocket-Chipを使ったRTLシミュレーションとFPGA実装

次に、RISC-Vのオープンソースによる CPU実装の手順を見てみよう。RISC-Vのオープンソース実装は現在では多くの企業や団体から公開されている。最も有名なものはカリフォルニア大学バークレー校や米SiFive社が管理している「Rocket-Chip (https://github.com/freechipsproject/rocket-chip)」だろう。Rocket-ChipはRTLシミュレーション、FPGA実装、はてはAWS F1インスタンスでの動作と、様々な環境をサポートしている。

このRocket-Chipの特徴としては、

- ソースコードがすべて公開されており、無料
- RISC-Vの仕様決定に近いメンバーが開発しているので、最新仕様にかなり追従している。
- 開発の経緯がすべてGitHub上で公開されており、現在でもアップデートが続いており開発が活発。
- ソースコードはVerilogやVHDLではなくChiselと呼ばれるハードウェア開発言語で記述されている。

おそらく多くの技術者にとってみれば、このChiselという言語のハードルが最も高いであろうと思われる。しかし、Rocket-Chipをダウンロードしてそのまま使うのであれば、Chiselの知識は必要ない。全自動でVerilogに変換されRTLシミュレーションからFPGAの構築まで行うことができるので、ユーザーはChiselの知識は全く必要ないのである(ただし改造しようとなるとChiselの勉強をしっかり行わないといけない)。

## Rocket-Chipのダウンロードとビルド

Rocket-Chipのダウンロードとビルドは、以下のように行う。(コード4)

Rocket-ChipをRTLシミュレーションする手段として、無料のRTLシミュレーション・ツールである「Verilator」を使用するか、米Synopsys社の有料RTLシミュレーターである「VCS」を使う方法の2つが提供されている。使用するツールに応じてディレクトリーが異なるので、適切なディレクトリーに移動して「make」コマンドを実行する。すると現在の構成パラメーター(今回はDefaultConfigとした)に応じて

コード6

```
$ grep "\[1\] pc=" output/rv64ui-p-add.out
C0:       53763 [1] pc=[0000000824] W[r 0=0000000000000000][0] R[r 8=0000000000000000] R[r
0=0000000000000000] inst=[fe0408e3] beqz    s0, pc - 16
C0:       53764 [1] pc=[0000000814] W[r 8=0000000000000000][1] R[r 0=0000000000000000]
R[r20=0000000000000003] inst=[f1402473] csrr    s0, mhartid
C0:       53767 [1] pc=[0000000818] W[r 0=0000000000000100][0] R[r 0=0000000000000000] R[r
8=0000000000000000] inst=[10802023] sw      s0, 256(zero)
C0:       53773 [1] pc=[000000081c] W[r 0=0000000000000400][1] R[r 8=0000000000000000] R[r
0=0000000000000000] inst=[40044403] lbu     s0, 1024(s0)
C0:       53781 [1] pc=[0000000820] W[r 8=0000000000000000][1] R[r 8=0000000000000000] R[r
3=0000000000000003] inst=[00347413] andi    s0, s0, 3
C0:       53782 [1] pc=[0000000824] W[r 0=0000000000000000][0] R[r 8=0000000000000000] R[r
0=0000000000000000] inst=[fe0408e3] beqz    s0, pc - 16
C0:       53783 [1] pc=[0000000814] W[r 8=0000000000000000][1] R[r 0=0000000000000000]
R[r20=0000000000000003] inst=[f1402473] csrr    s0, mhartid
C0:       53786 [1] pc=[0000000818] W[r 0=0000000000000100][0] R[r 0=0000000000000000] R[r
8=0000000000000000] inst=[10802023] sw      s0, 256(zero)
C0:       53792 [1] pc=[000000081c] W[r 0=0000000000000400][1] R[r 8=0000000000000000] R[r
0=0000000000000000] inst=[40044403] lbu     s0, 1024(s0)
...
```

コード7

```
git clone https://github.com/ucb-bar/fpga-zynq.git --recurse-submodules
cd zedboard  # zedboard / zybo / zc706のどれかを選択する。
make
```

Verilogのコードが生成される。

実際にテストベンチを渡してRTLシミュレーターでシミュレーションするためには、以下のように実行する。以下はRISC-Vのテストパターンである「rv64ui-p-add（加算命令のテスト）」を実行した結果である。

（コード5）

RTLの実行トレースが出力されているので、見易いように加工する。プログラムカウンターとアップデートされたレジスタファイルの情報などが記録されており、テストベンチの実行結果を確認できる。（コード6）

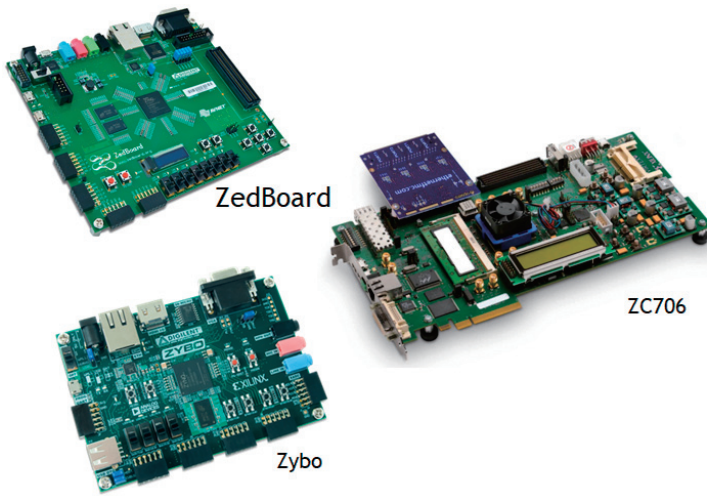



図1 RISC-Vの実装であるRocket-Chipは、Xilinx社のさまざまなボードでサポートされている

## ZedBoardを使用したRocket-ChipのFPGAインプリメント

Rocket-Chipは単体のCPU環境だが、これを米Xilinx社のFPGAにインプリメントするための環境も公開されている（図1）。「fpga-zynq」リポジトリー（https://github.com/ucb-bar/fpga-zynq）もしくは、「freedomプラットフォーム」リポジトリー（https://github.com/sifive/freedom）である。実はfpga-zynqリポジトリーはアップデートがほとんど行われていない。freedomプラットフォームならば最新に追従しているのだが、筆者はXilinx社の「ZYNQ」をメインに使用しているので、fpga-zynqを使用する。

Rocket-ChipでのZYNQ環境の構築は非常に単純で、fpga-zynqリポジトリーをダウンロードしてから（こちらもRocket-Chipをサブモジュールとしてダウンロードするためサイズが非常に大きいため要注

コード8

```
root@zynq:~# ./fesvr-zynq pk hello
hello!
```

コード9

```
root@zynq:~# mkdir /sdcard
root@zynq:~# mount /dev/mmcblk0p1 /sdcard
root@zynq:~# ./fesvr-zynq +disk=/sdcard/riscv/root.bin bbl /sdcard/riscv/vmlinux
```

意)、ビルドしたいFPGAボードのディレクトリーに移動してmakeを実行するだけである。これだけで、FPGAのブートに必要なFSBL、U-Boot、制御用Armコアのブートに必要な「PetaLinux」のビルドなどもすべて全自動で行ってくれる。(コード7)

しばらくするとビルドが完了する。Rocket-ChipのVerilogファイルと、Vivadoプロジェクトの生成、そしてFPGAに書き込むためのファイルの生成などすべて自動的に実行される。そして、生成されたビルドファイル群をSDカードに書き込み、ZYNQボード(今回はZedBoardを使用した)に差して電源を入れればよい。こうすることで、まずはArmコア上でPetaLinuxがブートされる。ZedBoardからの出力を観測するために、ZedBoardからUSBシリアルポートを接続する。今回は接続にTeraTermを使用し、図2のような設定を行った。

PetaLinuxにログインできたら、次はRocket-Chipを動作させる。Rocket-ChipはZYNQのPL部(ProgrammableLogic部)に書き込まれているため、PS部(ProcessingSystem部、つまりArmコア)から制御する必要がある。Armのコンソールから以下のように入力する。(コード8)

helloと表示されれば、Rocket-Chipが動作していることを確認できる。

さらに、フロントエンドからRocket-Chip上でLinuxを立ち上げることもできる。図3はRocket-Chip上でLinuxをブートさせた様子である。(コード9)

このように、RISC-Vを試してみたいと思ったとき、様々な手段が用意されている。特にFPGAでの実装は、好みのカスタマイズなどが容易なため、RISC-Vの導入を検討したい場合には一度試行してみるとよいだろう。

図2 Rocket-Chipの動作を観測するためのTeraTermの設定

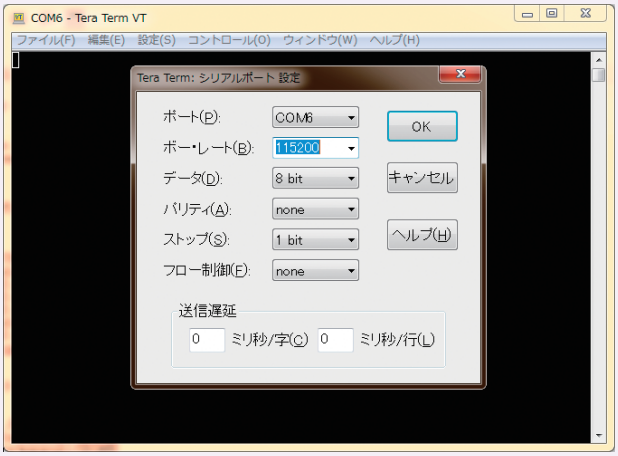
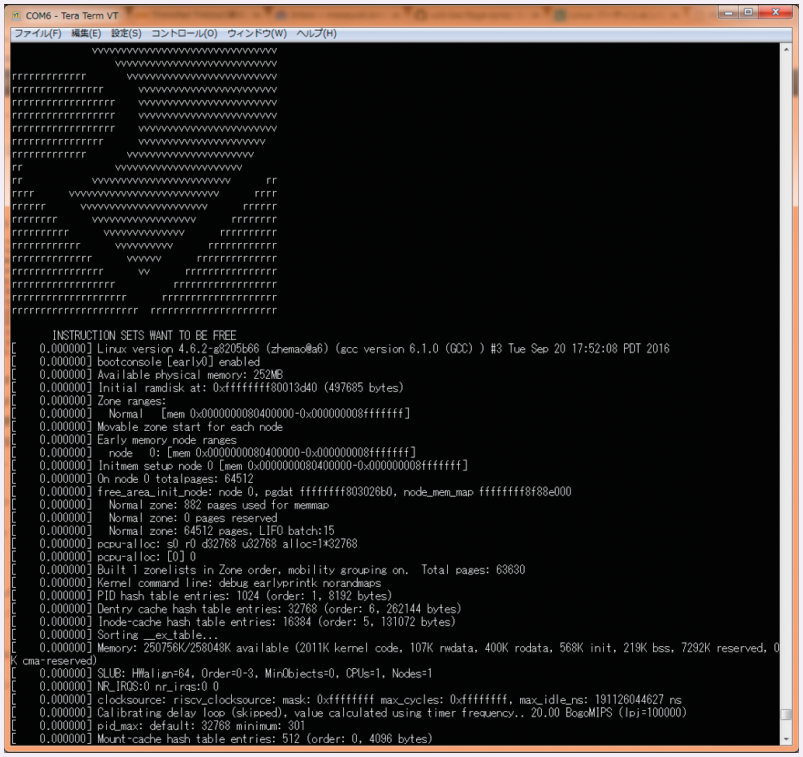



図3 Rocket-Chip上でLinuxをブートさせた様子



# TEEを中心とするCPUセキュリティーの動向

産業技術総合研究所　須崎 有康

スマートフォンでの指紋認証のような漏洩してはいけないデータの処理や、PCでリモートから機密情報を扱う処理を、OSとは独立してCPUで実行できる仕組みが求められている。
こうした要望を実現するため、最近のCPUには、OSとは独立して安全に実行できる環境を提供する機能「Trusted Execution Environment（TEE）」が搭載されるようになった。
現状で活用できるTEEとして、英Arm社の「TrustZone」と米Intel社の「SGX（Software Guard Extensions）」がある。
そして、オープンソース・アーキテクチャーとして普及が進むRISC-Vでも、TEEの機能を取り込む研究が進んできている。
ここでは、それに関係する技術動向を概観する。

TEEとは、OSとは独立して、プログラムを安全に実行できる環境のことである。その目的や実際の実装は、提供するサービスによって大きく異なる。誤解を恐れずにユースケースを定義すると、大きく二つに大別できる。一つは、スマートフォンなどのモバイルデバイスにおいて、デバイスが得た情報をOSとは独立して外部に出さずに処理するようなケース（例えば指紋認証）。もう一つは、PCなどで、外部から機密情報を含んだ処理（例えば暗号通貨に関わる処理）を依頼する際に、OSとは独立に処理するようなケースである。

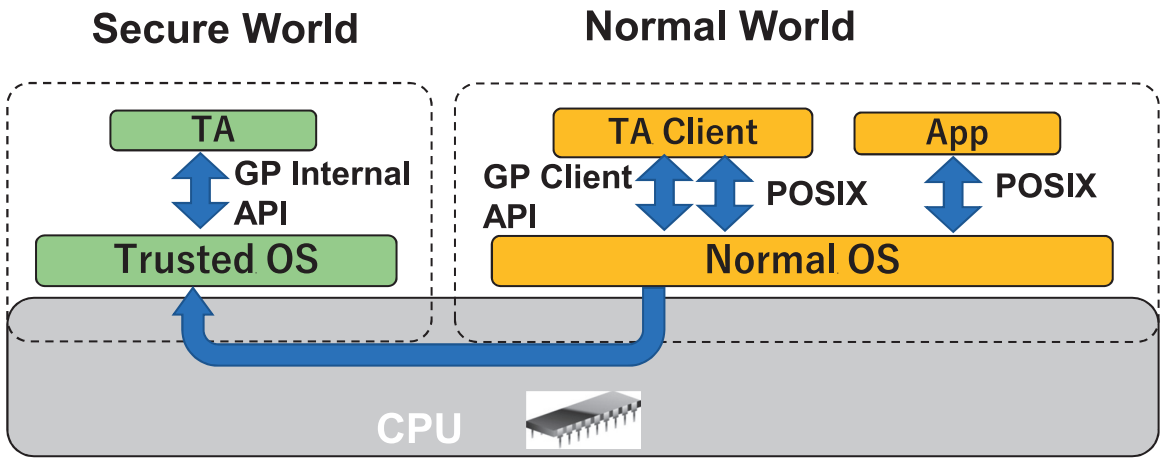
前者に対応するのがArm社のTrustZoneであり、電源投入時からTEE環境を用意する。ここには「Trusted OS」が実装され、その上で「Trusted Application（TA）」を動かすことが想定されている。後者に対応するのがIntel社のSGXであり、要求があるたびに独立した実行環境である「Enclave」を用意する。RISC-V向けについては、このEnclaveを発展させたTEEの研究が進んでいる。

## GlobalPlatfromのTEE要件と仕様

ICカードのセキュリティー仕様を定義している「GlobalPlatform」では、TEEとしての7つの要件を定義している。すなわち、①OSとは独立に実行すること、②それぞれのTAは独立して実行すること、③信頼された者のみがTrusted OSとTAを修正できること、④ブートプロセスがSoC（System on Chip）にバインドされ、TEEファームウエアとTAの真正性と完全性を強制できること、⑤信頼できるストレージを用意すること、⑥ペリフェラルには安全にアクセスすること、⑦最新の暗号化技術を使うこと、である。これらすべての要件を満たすのは、モバイルデバイス型のTEEである。

GlobalPlatformでは、CPUがNormal WorldとSecure Worldに分かれていることと想定しており、それぞれで通常のOSと

図1　GlobalPlatformが定義するTEEの概要図



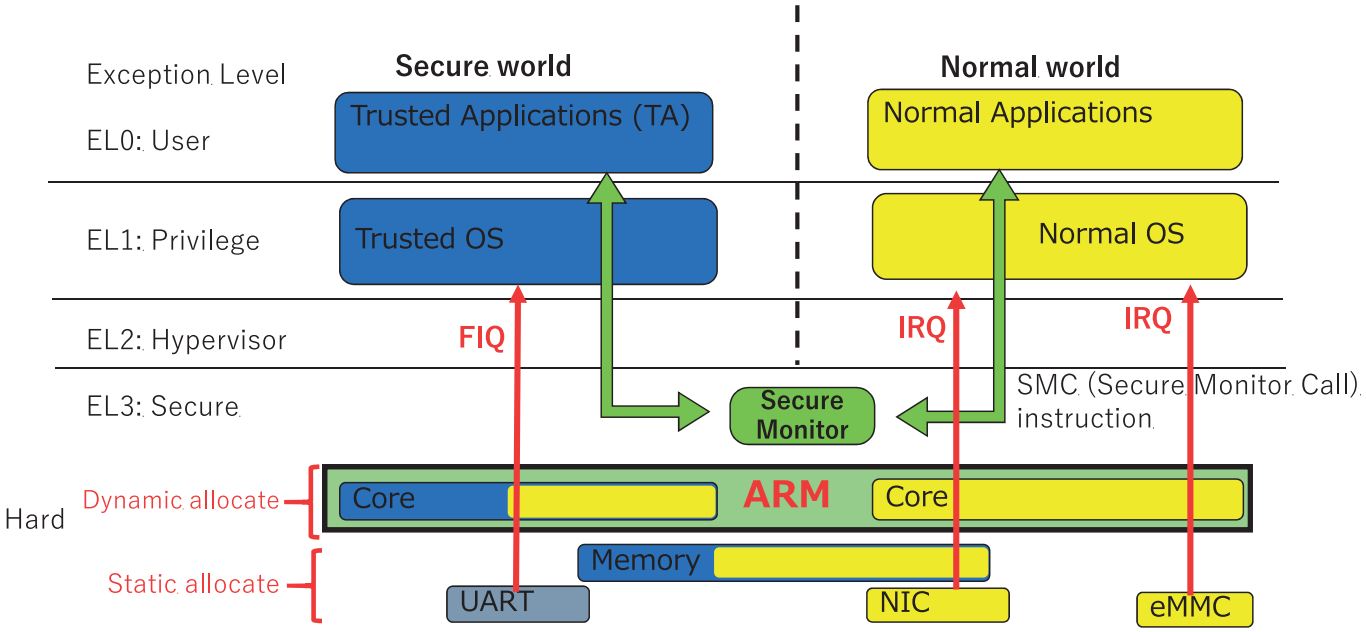



図2　Cortex-AのTrustZone実装概念図

Trusted OSを起動する。この様子を図1に示す。Trusted OSは通常のOSと異なり、ストレージやI/Oの機能がない。このためTAとして、通常のアプリケーションをそのままでは動かすことはできない。

Secure WorldのTAは、通常のOSでアプリケーションとして動くTA Clientからロード、起動、停止される。「TA Client」が使うAPIは、「GlobalPlatform Client API」として定義されており、TAがTrusted OSとやり取りを行うAPIは「GlobalPlatform Internal Core API」で定義されている。Internal Core APIはPOSIXと大きく異なり、暗号化のためのBig Integerなどを扱う。

## Arm TrustZone

Armコアは、モバイルデバイス型のTEEを代表するCPUコアである。GlobalPlatformの定義と同様に、CPUをSecure WorldとNormal Worldに分割して、通常のOSとTrusted OSを起動する。

Arm社のTrustZoneの歴史は古く、2003年の「ARMv6K」から導入されている。ただし、実際に使わるようになったのは「ARMv7」「ARMv8」からである。また、スマートフォンで使われている高性能の「Cortex-A」と、組込みで使われてCPUパワーに対する高い応答性が要求される「Cortex-M」ではアーキテクチャーが異なる。ここではCortex-Aを中心にTrustZoneを説明する。

Cortex-AでのTrustZoneの実装を図2に示す。Cortex-Aでは特権レベル(Exception Level)が4つあり、Secure WorldとNormal Worldそれぞれでこれら4つを利用してOSを動かすことができる。Normal WorldとSecure Worldの双方を行き来する際にはSMC命令を実行し、Secure Monitorを通して通信する。Secure Worldに割り当てられるハードウェアによってメモリーやペリフェラルが起動時に決まるが、CPUコアはSMC命令でモードが変わるので動的に役割が変わる。また、Cortex-Aでは、割込みをSecure WorldとNormal Worldそれぞれに割り当て、「Fast Interrupt reQuest(FIQ)」または「Interrupt ReQuest(IRQ)」として割込みを受けつける。

Cortex-AのTrustZone起動手順を図3に示す。図中のBLはBootLoaderのレベルを表しており、BL1からBL33までである。電源投入直後に実行されるBL1はBootROMでSoC固有のものであり、Secure WorldのEL3(Secure)で処理される。BL1ではBL2である「Trusted Boot Firmware」を検出して、EL1で制御を渡す。BL2ではSecure Worldで使うメモリーの確保やペリフェラルの割り当てを行う。その後はBL31をSecurWorldのEL3で「Secure Monitor」を、Secure WorldのEL1でSecure OSを、Normal WorldのEL1でNormal OSをそれぞれ起動し、定

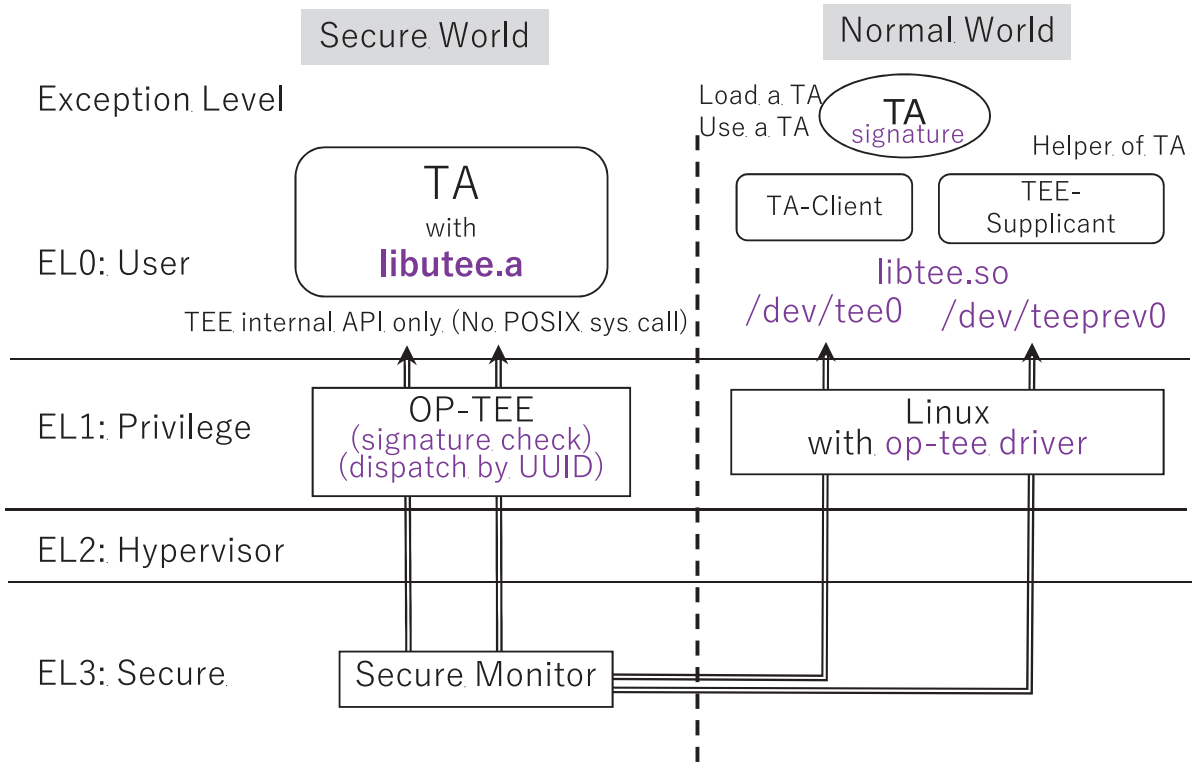



図3 Cortex-AのTrustZone起動手順

常状態へ移行する。

## Trusted OS

Cortex-AのTrustZoneでは、いくつかのTrusted OSが使われている。商用としては「iPhone」の「Secure Enclave」、韓国Samsung Electronics社の「Knox」、米Qualcomm社の「QTEE」、中国Huawei社の「TrustedCore」などがある。オープンソースとしては、フィンランのアールト大学の「Open-TEE」、米Google社の「Trusty」、Arm用Linuxカーネルを開発する非営利組織であるLinaroの「OP-TEE」などがある。ここでは、我々が使っているOP-TEEを基に説明する。

図4にOP-TEEの構成図を示す。OP-TEEではGlobalPlatformが定義するClient APIとInternal core APIのそれぞれに基づくAPIを提供しており、図中のTA ClientをTAがそれぞれ従わせる。TA ClientはLinuxの/dev/tee0を通してTAのロード、起動、終了を管理する。ロードするTAにはOP-TEEの秘密鍵で署名された「signature」が付けられており、これをOP-TEEが検証することでロードする。また、ロードする際には「UUID」が付けられ、このUUIDでTAを識別する。TAからI/Oの要求などは、「/dev/teeprev0」を通して、Linux上の「TEE-Supplicant」に要求する。

## Arm TrustZoneの脆弱性

Arm TrustZoneについての脆弱性はいくつか報告されているが、多くはTrusted OSおよびTAの実装上の脆弱性である。このうち、NDSS2018で発表された「Boomerang Flaw」と呼ばれる脆弱性は、多くの実装で生じる。Boomerang-Flawは、TAのポイント操作を悪用する攻撃であり、TrustZoneがSecure Worldの任意のメモリー空間にアクセスできることを利用する。攻撃されたTAから、Normal Worldで機密情報を含むメモリー空間にアクセスして情報を盗む。そのTAから攻撃者のNormal Worldのアプリケーションに伝えるため、Boomerang-Flawと呼ばれる。

## Intel SGX

SGXはIntel CPUで2015年にリリースされた「Skylake」から追加されたセキュリティー機能である。先に説明したように、外部からの処理依頼をOSとは独立に行うためのハードウエア機構である。SGXは

「Enclave」と呼ぶ隔離された実行環境を動的に提供するが、ここでの実行レベルはユーザーレベルである「Ring3」のみである。メモリーもEnclave専用の物理メモリー空間が割り当てられ、この領域はOSからアクサスされることはない。OSを提供するための異なる特権レベルはなく、基本的にOSをEnclaveで実行することはできない。また、Arm TrustZoneと異なり、ペリフェラルを使うことができない。割込みが上がった場合はEnclaveの実行を中断してOSの割込みハンドラを実行した後にEnclaveに戻ってくる機構がある。

Intel社では、SGXで実行するバイナリーが正規にデバイス上で動作することを確認する「Remote Attestation」機能を提供している。Remote Attestationにより、デバイス上のBIOS/UEFIが最新であるか、enclave内で実行しようとしているバイナリーが意図したものであるかを外部から確認することができる。

SGXを使った代表的なターゲットとしては、コンテナの安全な実行するが挙げられる。代表的なプロジェクトとしてはSCONEがあり、オープンソースとして提供されている。

## RISC-V Sanctum/Keystone

RISC-Vに向けたTEEには、Intel SGXを発展させたマサチューセッツ工科大学の「Sanctum」とカリフォルニア大学バークレイ校の「Keystone」がある。いずれもIntel SGXと同様にEnclaveを動的に作成できるが、特権レベルが1つではなく、複数に分かれており、Trusted OSとTAの分離ができる構造になっているが、まだ実装例がない。

Keystoneは2018年12月のRISC-V Summitにおいてオープンソース提供がアナウンスされた。現在の利用方法は3通り（QEMU、FPGA、RISC-V board）ある。QEMUエミュレーターで利用可能であり、RISC-Vハードウエアがなくても、誰でも動作を確認することができる。筆者は試していないが、FPGAあるいは米Amazon.com社のFPGAが使える「AWS」でも実行できる。また、米SiFive社が販売している「RISC-V unleashed ボード」上では実際にサンプルが動作することを筆者が確認している。

## RISC-V TEE WG

Keystoneは大学発の実験的なTEE拡張であるが、RISC-Vの仕様を決めるRISC-V FoundationでもTEEの仕様策定が進んでいる。これを進めているのはSecurity Chapterの下にあるTEE Working Groupであり、米NVIDIA社のJoe Xi氏がチェアを務めている。まだ、仕様は確定していないが、ペリフェラルの分離も想定されており、モバイルデバイス型のTEEになる予定である。

**[謝辞]** 本研究成果の一部は、NEDO「セキュアオープンアーキテクチャ基盤技術とそのAIエッジ応用研究開発」の支援によって実施したものである。

図4 OP-TEEの構成図