

自動車分野での半導体とソフトウェア技術の動向

JASA / Linux Foundation 共同 Webinar

宗像尚郎

hisao.munakata.vt(at)renesas.com

ハイパフォーマンスコンピューティング プロダクトグループ
HPC SoC ビジネス本部, SoC ソフトウエアイネーブルメント部
シニアダイレクタ

2025-3-12

自動車に HPC が搭載される？

HPC = 元々は 大規模並列コンピュータ (スーパーコンピュータ)



The screenshot shows the Intel Japan website with a search bar and navigation menu. The main content area features a video player showing a person working at a computer workstation in a data center. To the right of the video is a text box with the title 'High Performance Computing (HPC) とは何か?' and a sub-section header 'High Performance Computing が重要な理由'.

High Performance Computing が重要な理由

High Performance Computing (HPC) は、新しいものではありません。HPC ワークステーションとスーパーコンピュータは、長年にわたり学術研究において重要な役割を果たし、複雑な問題を解決し、発見と革新を促進してきました。

科学者、エンジニア、研究者は、天気予報、石油・ガス探査、物理学、量子力学、およびその他の学術研究および商用アプリケーションの分野での幅広いユースケースで、HPC を利用しています。

HPC の並列コンピューティング機能は、従来のコンピューティングと比較して反復プロセスを大幅に加速できます。例えば、HPC は、ディープラーニング・モデルのトレーニングにかかる時間を数日から数時間に短縮できます。AI とビッグデータ・アプリケーションが普及し、高度なコンピューティング・リソースがより利用しやすく手ごろな価格になるにつれて、HPC はますます広範な課題を解決するために導入され、幅広いイノベーションを実現しています。

<https://www.intel.co.jp/content/www/jp/ja/learn/what-is-hpc.html>

第 5 世代 R-Car SoC の X5H をプレスリリース (2024.11.13)

製品情報

アプリケーション

設計リソース

サポート

サンプル&ご購入

会社情報

会社案内 > ニュースルーム > 第5世代R-Carの第一弾、3nmプロセス採用の車載用マルチドメインSoC「R-Car X5H」を発表

第5世代R-Carの第一弾、3nmプロセス採用の車載用マルチドメインSoC「R-Car X5H」を発表

～チップレット技術の適用により性能拡張が可能なハイエンドセントラルコンピューティングソリューション～

2024年11月13日



車載用マルチドメインSoC R-Car X5Hを発表

ルネサス エレクトロニクス株式会社（以下ルネサス）は、このたび、第5世代R-Car（R-Car Gen 5）の第一弾製品として、ADAS（先進運転支援システム）、IVI（車載インフォテインメント）、ゲートウェイの複数のアプリケーションに使用できる新世代の車載用SoC（System on Chip）「R-Car X5H」を発表しました。業界最高レベルの高性能を実現すると同時に、最先端の車載用3nm（ナノメートル）プロセスを採用した高集積化により低消費電力化を実現しました。ユーザは、車両のセントラルコンピューティングECU（電子制御ユニット）に新製品を使用することで、将来を見据えたシステム開発を効率的に実現可能です。R-Car X5Hは、2025年上期に一部の自動車顧客向けにサンプル出荷を開始し、2027年下期に量産を開始する予定です。

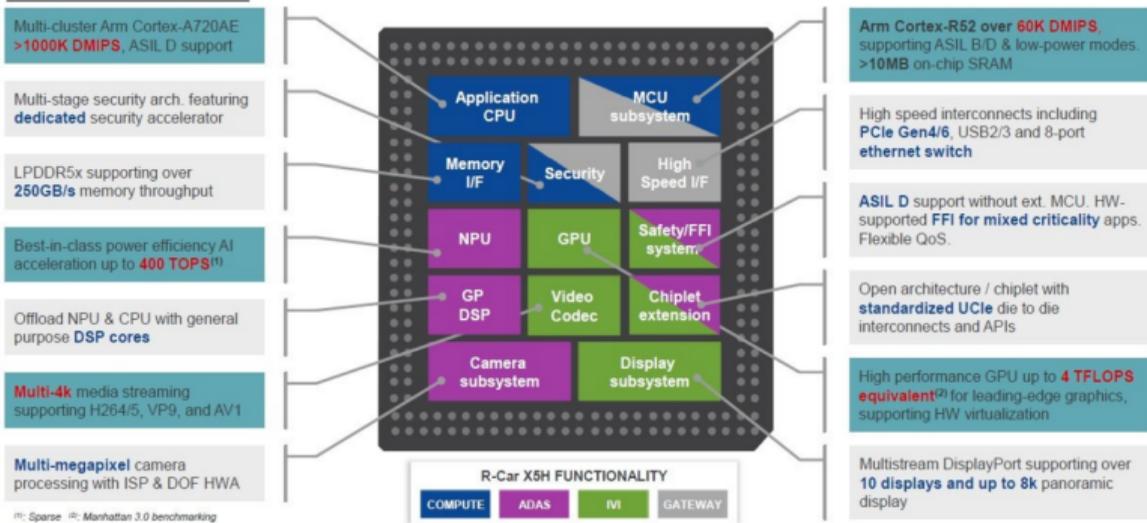
R-Car X5Hは、アプリケーション処理用に32個のArm® Cortex®-A720AE CPUコアを搭載し、1,000k DMIPS以上の性能を発揮します。また、6個のCortex-R52 CPUコア（ロックステップ）で60k DMIPS以上の性能を実現し、外付けマイコン無しでASIL Dを実現可能です。さらに、最大400 TOPS（Sparse）のAIアクセラレータと、最大4 TFLOPS¹のGPU（Graphics Processing Unit）を搭載しました。チップレット技術を採用したことにより、AI性能やグラフィックス処理性能を拡張することも可能です。

<https://shorturl.at/nmfCq>

次世代の 自動車の中央制御 に必要となる HPC 機能を 1chip に集約

R-Car X5H SoC: SCALABILITY

Designed to address the shift to centralized architecture



車載 HPC (=R-CarX5H) は サーバー級の計算パワー を持っている

R-Car X5H SoC		Intel Core i9-14900K (Raptor Lake Refresh)
Arm CA9 x 32 Arm CR52 x 6 (dual LS x 3)	CPU core number	24
1,000k (CA) + 60k (CR)	CPU (DMIPS)	450k ~ 900k *1 (PassMark 45,000)
2,000 (support 8k x 10 input)	GPU (GFLOPS)	793
400	NPU (TOPS)	Not included
2.7GHz	CPU clock	3.2 GHz
3 nm	process node	10 nm
2,916	pin count	1,700

1: We used 1 PassMark = 10-20 DMIPS as a general approximation performance conversion

HPC の SW 開発では、アーキテクチャ・開発方法・保守など全てを変える必要がある

自動車に HPC が搭載される？

自動車向けの SW 開発

SW 開発手法のイノベーションへの対応

これまでの自動車向けの SW 開発の実像
今、世界の自動車産業でおこっていること

自動車向けの SW 開発

これまで 自動車のソフトウェアは「組み込み SW」として開発された

自動車の制御では さまざまな制約条件下で「機能安全」を達成 する必要がある

- CPU の処理能力が限られている（時間制約）
 - 割り込みイベントに対する応答時間が規定されている（リアルタイム性）
 - 起動時間の制約（スタートアップ処理の軽量化）
 - 利用可能な電力の制約（発熱対策、電池駆動時間の確保など）
- メモリーサイズを変更（拡張）できない（容量制約）
 - RAM サイズ = ワークメモリー、フレームバッファ（画面サイズ）
 - ROM サイズ = プログラム容量の制約
- 無謬性（バグがひとつも無いことが求められる）
 - 原則、製品出荷後には SW を変更ができない（信頼性要求レベル は極めて高い）
 - ユースケースに基づいたシステム検証 によって十分なテストカバレッジを確保する

組み込み機器の制御用 SW では「多くの制約を回避する繊細な制御」が必要になる

「完全予定調和」を作り込むことで「機能安全」を実現してきた

「完全予定調和 (determinacy)」に求められる機能

- タスクの実行順序を完全に管理する → **タスク優先度** による実行順序の制御
- 割り込み応答時間の保証 → **割り込み禁止区間** の設定による応答時間の管理
- メモリー管理 → **静的なメモリー割り付け**
- HW 資源管理 → **静的な機能割り付け** (特定アプリに HW 資源を占有させる)
- 障害解析、リカバリー → **ログメッセージ埋め込み、エラーハンドル処理** の実装
- 組み込みシステムでは **SW 設計者が「システム全体の挙動を完全に把握」** し、想定されるあらゆる状況において **システムが破綻しない** ようにソフトウェアを設計し **「完全な予定調和」** を実現させることに **全面的な責任を負っている**

組み込み環境では HW/SW 構成が変化しなければ「完全予定調和」も実現可能である

RTOS を積極活用 することで「完全予定調和」を実現している

プログラマーがタスクの実行を完全にコントロールできる

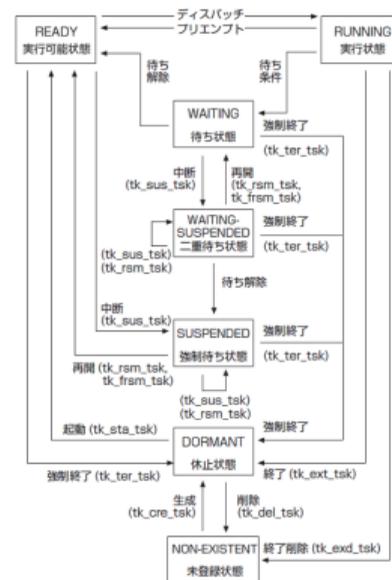
■ プリエンプション (preemption)

- 優先度の高いタスクが 実行中のタスクに割り込む
- プログラムからタスクを起床できるのでプログラマーの想定通り (deterministic) に 処理を記述できる
- 割り込み処理ルーチンから直接タスクを起床できるので 割り込み発生から実行までの時間を管理 できる

■ プログラマーは 複数タスク間の状態遷移 (= 予定調和の維持) について全責任を持つ (マニュアル調停)

■ 別タスク (新アプリ) を追加したら予定調和は破綻

図 1. タスク状態遷移図



T-kernel のタスクの状態遷移

https://www.zion.org/wp-content/themes/dp-magiam/pdft-kernel_2.0/html_tajtask_states_and_scheduling_rules.html

ユーザーは 自動車にも「スマホ的なユーザー体験」を求め始めた

古典的な組み込み開発（例：旧自動車）

- ハードウェア仕様が先に決定される
- ベンダー固有機能の利用も躊躇しない（非標準 API を持つアクセラレータ等）
- 毎回スクラッチ開発（再利用しない）
- 各チップ専用の開発環境
 - IP 専用の SW ライブラリー
 - BSP ← ボード専用の SW 環境
- ソフトには 完全予定調和 が求められる
自動車では 機能安全 も要求される

スマートフォン開発（SW First 指向）

- 特定の HW に紐づいていない
- 自動調停で幅広い HW に適合できる
 - プロセッサ性能
 - メモリー、ストレージサイズ
 - 画面サイズ、横長/縦長
- アプリは複数が同時並行開発される
- HW と SW の進化は非同期となる
- アプリ間の実行調停は OS に委ねる
- 汎用高機能 OS 抜きには実現できない

この2つの開発の考え方は根本的に異なっており、相容れないと認識する必要がある

スマートフォンは「組み込み機器」とは本質的に異なっている

スマートフォンは「汎用 OS の自動調停機能」がなければ成立しない商品 である

- 「ユーザーによるアプリインストール」、「SW の逐次更新」はスマートフォンの本質なので、開発者による SW の予定調和は意図的に断念した。その代わりに「汎用 OS 内蔵の自動調停機能」に委ねることで「SW First の世界観」を実現
- 「汎用 OS 内蔵の自動調停機能」
 - タスク スケジューリングの 動的な自動最適化機構
 - 割り込み処理の最適化
 - 投機的なメモリー利用（空きメモリーはソフトキャッシュ機構に全面活用）
 - リソースの動的最適アロケーション、プロセス間共有（メモリー、HW 資源）
 - システム プロファイリング、デバッグ支援機能

スマホの信頼性担保の拠り所は、組み込み機器の「ガラケー」とは全く異なっている

「RTOS」と「汎用 OS (Linux)」の基本的な制御戦略の違い

RTOS = 設計者が完全予定調和を作る

- RTOS の適用対象
 - 白物家電、ガラケー、産業機器
 - 自動車の ECU 制御用
- マイコン向け (基本はシングルコア)
- リアルタイム性が担保 される
- 機能安全 に対応した商用製品もある
- OSS と 商用製品 がある
 - OSS : FreeRTOS、Zephyr など
 - 商用 : SafeRTOS、QNX など

Linux = OS の動的な自動調停に委ねる

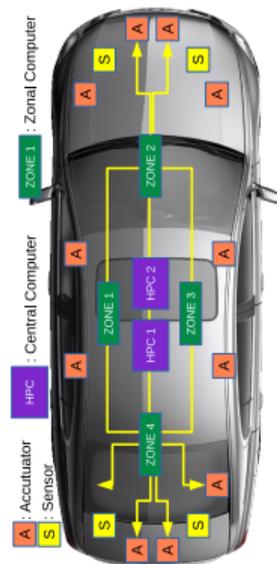
- Linux の適用対象 (非常に幅広い)
 - スーパーコンピュータ (科学、医療)
 - サーバー (金融、証券)
 - 車載 (IVI)、Android スマートフォン
 - IoT、ルーター
- 1,000 コア、5,000 スレッドに対応済
- マルチユーザー、マルチタスク対応
- メモリー保護、仮想化 による分離
- オープンソース ← 挙動解析が可能

RTOS と Linux では「タスクの実行順序決定のアルゴリズム」が大きく異なってくる

100 年超の自動車の歴史 で初の「コンピュータアーキテクチャ革命」

新世代車載アーキテクチャ = Vehicle Computer (HPC)

- インフォテイメント (HMI)
- ゲートウェイ (ネットワーク ルーター)
- Vehicle Computer (HPC) というアーキテクチャ革命
- HPC は専用コンピュータ (=組み込み機器) ではない
- さまざまなワークロードへの対応 が可能になる
 - 従来、専用 ECU で実現していた 複数の機能の集約
 - リアルタイム性の保証が必要なユースケース もある
 - 機能安全要件 への対応が必要な部分も出てくる
 - クラウド連携 や サイバーセキュリティ対策 への考慮も必要



「一機能一 ECU」に分散していた処理を HPC へ集約し、更に機能の部分更新を目指す

Vehicle Computer (HPC) 時代の自動車制御 に求められるもの

固い部分：安全性、予測可能性、機能安全

- **FFI (Freedom From Interference)**
 - 統合するプログラムの独立性の担保
 - (広義の) リアルタイム性の保証
 - (部分的な) 静的資源割り当て
- **コプロセッサへのオフロード**
 - ヘテロ構成 (専用コア)
 - 疎結合 = 並列実行性
 - 時間分離 (リアルタイム性)
 - 空間分離 (機能安全)

柔らかい部分：柔軟性、部分更新対応

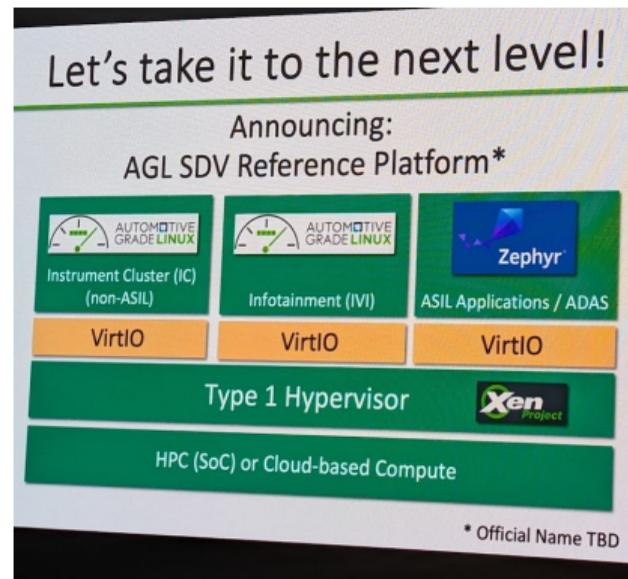
- **SW の部分更新への対応**
 - スマフォライクなユーザー体験
 - 保守性 (サイバーセキュリティ対応)
- **仮想化技術の導入**
 - コンテナによるアプリ分離
 - ハイパーバイザーによる HW 分離
- **SW 技術標準化への対応**
 - オープンソース (OSS) の利活用
 - 産業コンソーシアムとの連携

デジタル時代の自動車の制御では「固い部分」と「柔らかい部分」の共存が必然となる

HPC は OSS で成立 している、自動車に OSS が波及する のは必然？

自動車に適用可能なオープンソース ソフトウェア

- OS (Linux)
 - AGL (IVI、クラスター、他)
 - ELISA (機能安全対応)
- RTOS
 - Zephyr
- HV (Type1)
 - Xen
- HV / VM (仮想マシン) 間インターフェース
 - virtio



Linux Foundation 傘下には自動車向け HPC で利用可能な OSS 資産が多い

SW 開発手法のイノベーションへの対応

自動車業界では 何故これまで SW の再利用が進まなかった のか？

現実には 多くの開発現場でスクラッチ開発 になっているのではないかな？

- 「完全予定調和」追求の弊害
 - 特定の HW/SW 構成を前提に 過度な最適化 が行われている
 - コードサイズ が大きすぎる（大きな塊になっていて、因数分解が不可能）
 - コード 難読性（コメントが無い、改訂履歴に説明がない等）
 - コード アクセス（コードを検索する方法が無い、コードが集約されていない）
- HW 依存性
 - 特定の HW 構成（アドレス、CH 数等）が 暗黙の前提 になっている
 - 汎用機能と専用機能（アクセラレータ等）が 分離 されていない
 - 制御対象ボード（≡ 仕向け、グレード）別に、類似のコードが乱立している
- NDA 制約（発注者によってコードの開示範囲が制限されている）

これまでは「コード再利用のコスト」がメリットを上回ってしまうケースが多かった

オープンソースは「再利用を前提として」SW を開発している

オープンソース (= OSS) 開発の流儀

- **Release early, release often** (機能実装完ではなく、時間ベースのリリース)
 - 決められた周期 (9 週、四半期など) でエンドレスにリリース を繰り返す
 - 結果として 締め切り間際の dirty な修正の混入を回避 している
 - トレース可能な 「小さな変更」を「多数」集積 する
- 完全な **透明性 (Transparency)** と記録の保全
 - **公開 ML** 上での議論 (メールアーカイブのデータベース化)
 - **公開 git** 上で変更来歴を管理 (コミットメッセージのフォーマットが規定されている)
- プロジェクトの **スケーラビリティ** (大規模開発に対応)
 - 全世界で数千名の開発者が参加する **多拠点同時分散開発**
 - **権限移譲** =カスケードされた Sub-system メンテナー体制

自動車業界でも OSS 開発プロジェクトで既に確立されたメソッドを参考にするべき

オープンソースは「集合知の結集」なので「生成 AI」との相性が抜群

膨大なクラッシュログから 瞬時に原因を特定 できる

■ カーネル起動ログ の調査

- 今後障害の要因となる **懸念事項の指摘**
- **性能改善** の可能性がある修正を提案
- **ワーニングを減らす** ための提案

■ クラッシュログ の解析と対策の提案

- クラッシュ原因の特定（複数要因の提示）
- 詳細な対策案の提案
- 実際の修正作業の支援

■ ソースコード生成

- スクリプトの作成、試験実行、改善提案
- ドライバーやアプリのコード生成
- **実験用、勉強用の教材としては十分 利用可能**

添付は Ubuntu 24.04 Server がクラッシュした時のログです（クラッシュする直前から記録しています） 何が原因でクラッシュしたかわかりますか？

Ubuntu 24.04 Server のクラッシュの原因は、カーネルの NULL ポインタ デリファレンスによるクラッシュであり、RAID 1 関連のコード (sync_request_write in raid1 module) で発生しています。

重要なログのポイント:

1. クラッシュのトリガー

```

yamt 03月 04 09:23:58 kgb2 kernel: BUG: kernel NULL pointer dereference, address: 0000000000000000

```

- カーネルの NULL ポインタ デリファレンス (#697: supervisor read access in kernel mode) が発生
- sync_request_write+0x7b/0x100 [raid1] で発生 (RAID 1 の同期処理)

2. 関係するプロセス

```

yamt 03月 04 09:23:58 kgb2 kernel: CPU: 11 PID: 425 Comm: md0_raid1

```

- md0_raid1 がクラッシュの引き金になったプロセス

3. RAID 1 の異常

```

yamt 03月 04 09:23:58 kgb2 kernel: RAID1: sync_request_write で NULL ポインタ デリファレンス

```

- RAID 1 の書き込み処理中にクラッシュ

4. SMART 警告

```

pgoop 03月 04 09:43:13 kgb2 smartd[1256]: Device /dev/sda [SAT], SMART Prefailure Attribute: 1
03月 04 09:43:13 kgb2 smartd[1256]: Device: /dev/sda [SAT], SMART Usage Attribute: 198 Air

```

「デジュール スタンドダード」から「デファクト スタンドダード」の潮流

IT 業界では「リーダーシップ企業」が戦略的に「デファクト化」を推進している

■ デジュール (dejure = 原則上の) スタンドダード

- 公的な **技術標準化団体** で、あらかじめ決められた手順を踏んで方式を策定
- 各企業の思惑などにより合意形成には **とても時間がかかる**
- Autosar、Jaspar、ISO/IEC、IEEE、JIS などの標準化団体がある

■ デファクト (de facto = 事実上の) スタンドダード

- **多くのユーザーが選択したものが結果的に業界標準になる**
- 企業はマーケティング活動、プロモーション活動を通じてデファクトを取りにいく
- **AGL (Automotive Grade Linux)** は自動車向けコンポーネントを集約して提供中
- **Android Automotive OS (AAOS)** のように事実上の標準を目指しているものも

今後自動車業界でも IT 連携が進むのに合わせデファクト化を取る動きが活発化する

産業コンソーシアム活動は コミュニティ開発とは性質が異なる

OSS 開発コミュニティ = 個人ベース活動

- コミュニティの形成は自然発生的で、明確な創始者が居ない場合もある
- 活動参加は任意（通常、契約は不要）
- 原則年会費などの費用負担は無い
- 積極的な貢献が期待される
- 規模が大きくなると、コードリリースサイクル管理、マージルール等の運用ルールの整備が必要になる

産業コンソーシアム = 企業ベース活動

- 創設メンバー (Founding Member) (多くの場合企業) が明確である
- 明文化されたメンバーシップ契約に基づいて活動に参加する
- 参加費や年会費などの義務がある
- 開発参加には CLA (Contributor License Agreement) が必要な場合も
- 開発リソースコミット (=FTE) の要請

産業コンソーシアム活動は、明確な目的や意思を持った企業による共創活動である

何故多くの企業が 共創開発（＝オープンな R&D 活動）に取り組むのか

多くの先端企業が「make or buy」から「make or buy or collaborate」に転換

- make 戦略 = 差異化技術については自社内でクローズに技術開発
- buy 戦略 = 非差異化領域の技術、時間が求められる技術は社外から技術導入
- collaborate 戦略 = 他社と連携して差異化技術のベースを共同開発
 - 開発規模が大きく自社だけでは必要なリソース（資金、開発者等）を集められない
 - オープンな活動とすることで、技術の中立性担保や開発スピードアップを図る
 - 企業間の連携による マジョリティ確保でデファクトを獲得する
 - 中長期的にメンテナンス可能な サステナブルなソリューションの構築
 - 企業同士で 似て非なる技術を重複開発するリスクを回避する

技術進化の速度、規模の拡大、グローバル化のトレンドを反映した技術の新潮流である

企業エンジニアがコミュニティに参加する時に 配慮すべきこと

会社から派遣された エンジニアが活動しやすい仕組み作り が必要

- 共創開発への参加に伴って発生する 色々な制約の分析と解決を図る 必要がある
 - CLA (Contributor License Agreement) 締結が求められる場合には契約条項の精査が必要、内容によっては参加できないケースもありうる
 - 公開 メーリングリストへの情報発信の制限解除 (タイムリーな応答が求められる)
 - コード投稿 (自社内の 社外技術発表ルールとの整合 が必要な場合も)
- オープンな開発に必要な 技術情報 (TRM、ボード回路図など) の公開
- 活動時間帯 の配慮 (国際組織の場合 オンライン会議が夜に開催される 事も多い)
- CLA、SW ライセンス など契約文書に対する法務部門の支援
- (番外) 個人開発者に対する開発委託契約 (=発注) に関するサポート
- 社外活動での悩み (課題) と成果を社内での共有 ⇒ 成果を適切に評価する

コンソーシアム活動内での 振る舞いの作法

自社利害だけを声高に主張するのは禁じ手 である

- コンソーシアムへ参加する各企業の代表は、自社の利益代表として振る舞うのではなく **コンソーシアム全体としての成果を上げることを第一義に考えるべき** である
- 特に参加企業が **自社内で開発した成果をコンソーシアムに持ち込む場合** には
 - 提供した技術が他のメンバー企業に使ってもらえるような **丁寧なサポート（ドキュメント等の整備、Q&A 対応等）** を行う
 - **開発ロードマップの共有** と、コンソーシアム内での **開発のリーダーシップ** を取る（コードの投げっぱなしはダメ、それでは何も起こらない）
 - サイバーセキュリティ対応など **必要な SW メンテナンス** を長期的に提供する
- 車輪の再発明を回避するために、**既存の類似の活動（OSS コミュニティ、他の産業コンソーシアム、技術標準化活動）との整合** を図るべきである

活動が成功するか否かはコンソーシアムのリーダーの指導力・調整力に大きく依存する

Linux Foundation の OSS トレーニング教材

OSS 技術を広範にカバー するコースが整備されている

- Linux システム運用（基本、応用）
- Linux カーネル開発（基本、応用）
- セキュアコーディング関連
- AI/ML 関連技術
- DevOps パイプライン構築
- クラウド・コンテナ活用
- OSS ベストプラクティス（コミュニティ参加支援）
- その他多数

The screenshot shows the 'Education' page of the Linux Foundation. The main heading is 'フルカタログ' (Full Catalog). Below it, there are filters for 'All' and 'Training'. A list of courses is displayed, including:

- Kubernetes Fundamentals (LFS258)** - \$299. Description: Kubernetesの複雑な概念を身に付けましょう.
- Kubernetes for Developers (LFD259)** - \$299. Description: マルチノード クラスターでアプリケーションをコンテナ化し、ホスト、サブドメイン、および構成する方法を学びます。
- DevOps & SRE Fundamentals: Implementing Continuous Delivery (LFS261)** - \$299. Description: CI / CDプロセスを自動化して、信頼を持ってソフトウェアをリリースするための準備を学びましょう。

On the left side, there are sections for 'プロダクトタイプ' (Product Type) and 'Areas of Interest' (Areas of Interest) with checkboxes for various topics like AI&ML, Blockchain, Cloud & Containers, etc.

今後の JASA/LF 連携プロジェクトの進展に大いに期待しましょう