OSAKA NDS Embedded Linux Cross Online Forum #11
～Hypervisorシステムのデバッグと可視化の検討～

2020.7.11
株式会社DTSインサイト
木村健太郎

自己紹介

株式会社　DTSインサイト
木村　健太郎

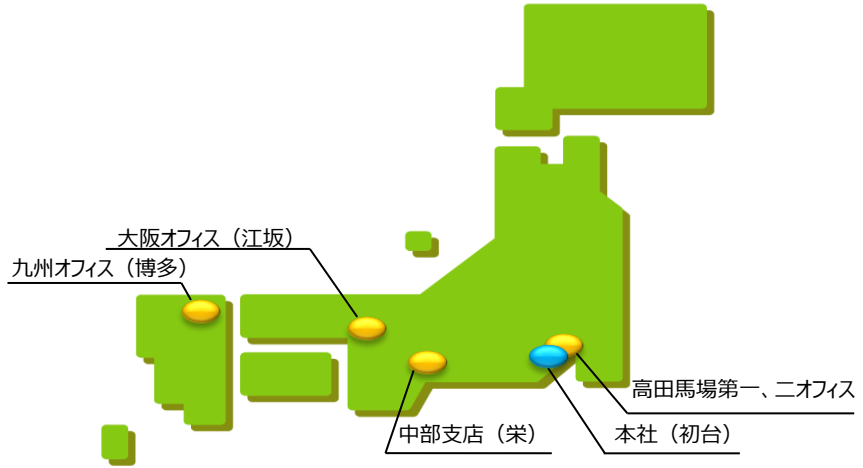
99年に合併前の横河ディジタルコンピュータに入社。
以来、adviceシリーズICEのデバッガ開発に従事。
　デバッガmicroVIEW-PLUSの開発初期メンバーとして参画以来、組込みOS対応、マルチコア対応、SMP対応、などデバッグ環境の進化に応じた設計を担当してきました。
　動的解析ツール、受託開発を経て、現在は古巣ICE開発にて、**新製品adviceXrossの開発**、Hypervisor環境のデバッグや可視化の対応を推進中。

https://www.dts-insight.co.jp/corporate/recruit/2019/interview03.html

会社概要

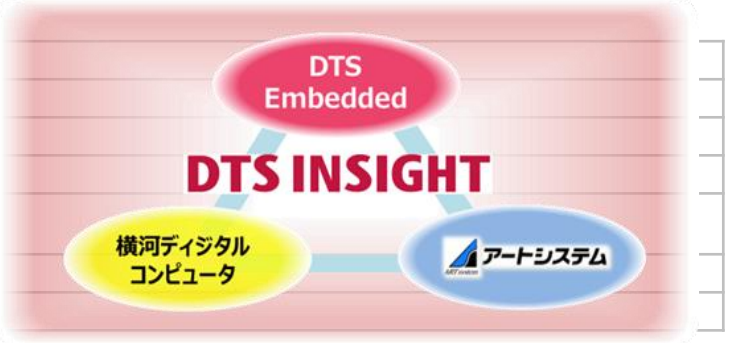
【会社概要】

商号	株式会社DTSインサイト （英文名：DTS INSIGHT CORPORATION）
設立	2001年6月
創立	1972年3月
資本金	2億円
従業員数	372名（2020年4月1日現在）
役員	代表取締役社長　浅見 伊佐夫 代表取締役常務　鴨林 英雄 取締役　安藤 裕一 取締役　浦島 邦明 取締役　中村 裕 監査役　赤松 謙一郎
拠点	東京（初台、高田馬場）、名古屋、大阪、福岡
関連会社	株式会社DTS他、DTSグループ各社



【会社沿革】

1972年	株式会社データ通信システム設立、デジタルコンピュータ株式会社設立
1980年	アートシステム株式会社設立
1990年	「デジタルコンピュータ株式会社」から「横河ディジタルコンピュータ株式会社」に社名変更
2003年	「株式会社データ通信システム」から「株式会社DTS」に社名変更
2014年	横河ディジタルコンピュータ株式会社が株式会社DTSの子会社に アートシステム株式会社が株式会社DTSの子会社に
2015年	アートシステム株式会社が株式会社DTSの組込み関連事業の一部を承継
2017年	横河ディジタルコンピュータ株式会社とアートシステム株式会社の両社が合併し、株式会社DTSインサイトを設立



事業内容

DTS GROUP

お客様

DTS INSIGHT

組込み開発ツール販売

エンジニアリング
サービス

車載

社会インフラ

医療

Factory Automation

今日お伝えしたい事

Hypervisorを採用/提案するにはデバッグと検証環境に課題が・・・（お客様）

- **仮想環境のデバッグ**

  HW上で直接実行する、Type1 Hypervisorのデバッグについて紹介

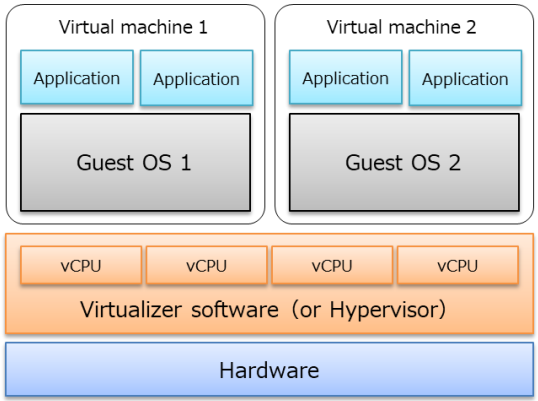
  複数のゲストOSがスケジューリングされるHVシステムをどのようにデバッグするか？

  - Armv8-AのVirtualizationと合わせて簡単に解説
  - デバッグ事例の紹介
    - Xen on Arm® にて、Xenのトラップコードを覗いてみる

- **仮想環境の可視化**

  可視化の目的はなにか？自分の処理は遅くないよね・・・？

  - バックエンドの時間を知る方法は？

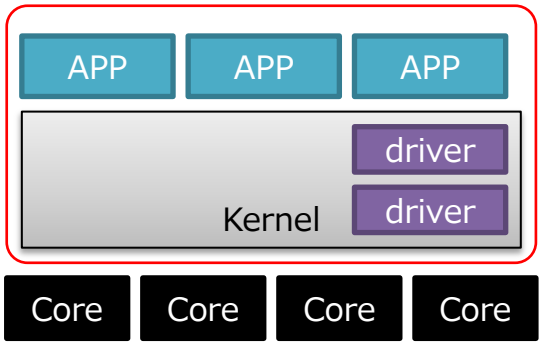


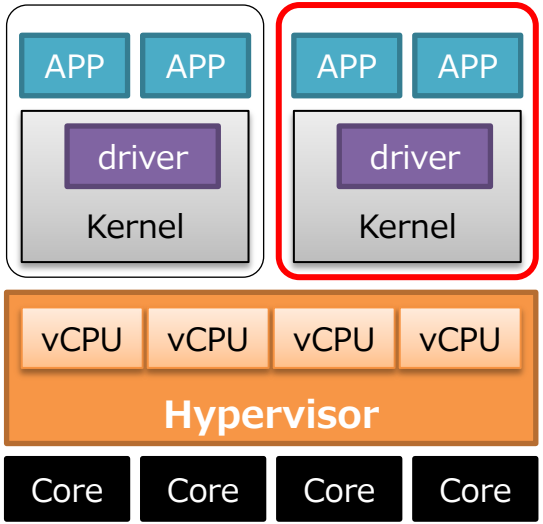
Type1 Bare metal

仮想環境のデバッグ

Hypervisorになってもやりたことって？

- OSを、ドライバを、アプリを、デバッグしたい！
  - 今まで同様、シングルOSのデバッグで操作したことと**同じこと**がしたい。
  - 欲張りなケースとして、念のため複数のVMを同時にデバッグできるといい。





解説！　AArch64　virtualization

contextの識別

- AArch64特権モデル（pstate.EL）
  - EL2＝ Hypervisor
- VMID（VTTBR_EL2.VMID）
  - 仮想マシンID＝Guest-OS

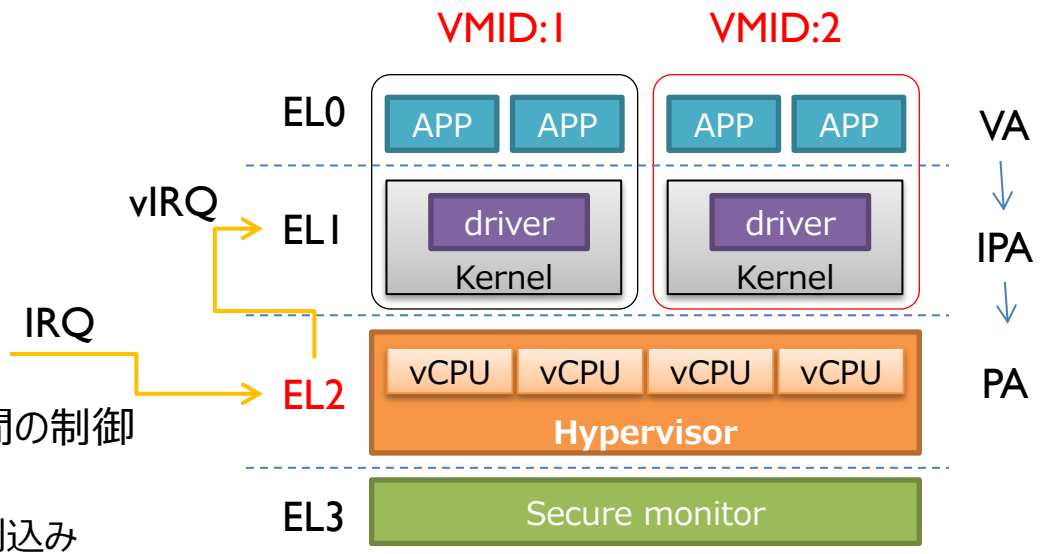
物理アドレス解決

- 2段階のメモリ変換
  - [VA] →[IPA] →[PA]

HVが直接制御

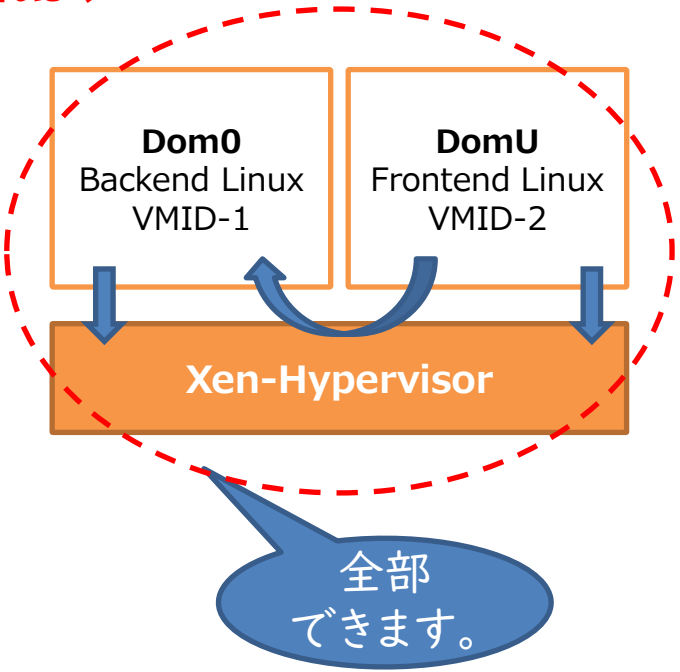
HV=物理リソースの割り当て、実行時間の制御

- 例外とルーティング
  - EL2で受け付けてからの→仮想割込み
- vCPUのスケジューリング



Hypervisor環境のデバッグシステムとは？

カレントを識別して、VM毎にデバッグシステムを切り替えれば、
いままでと同じデバッグができる



マルチOSデバッグ

- **「VMコンテキスト」**ごとのデバッグ操作を実現
- カレントのVMコンテキストを**自動検出**
- 「OS認識機能」と「OSシンボル」を**自動切換え**

VMIDフィルター機能

- 指定したVMコンテキストでブレーク（Watchpointブレーク）

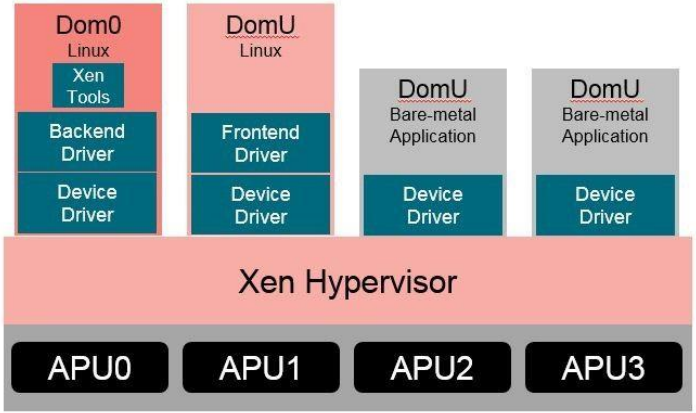
各種ARMv8仮想対応

- メモリの2段階アドレス変換

実際にデバッグをしてみた！「Xen on Arm」

- 特徴
  - オープンソース（だれでも手軽に利用可）
  - **Type1 Hypervisor**

- HWの仮想化サポートを可能な限り活用している
  - カーネルモードとハイパーバイザモードを切り替えるために導入されたHVC命令
  - MMUは２段階のページ変換をサポート
  - 汎用タイマー、GIC割込みコントローラは仮想化に対応



出典：https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842530/XEN+Hypervisor

実際にデバッグをしてみた！「Xen on Arm」

Xenの実行環境

ICE　　：adviceXross 新製品！

SoC　　：Xilinx社 Zynq UltraScale+ MPSoC（Cortex-A53x4）

Boards ：Xilinx社 ZCU102/ZCU106ボード

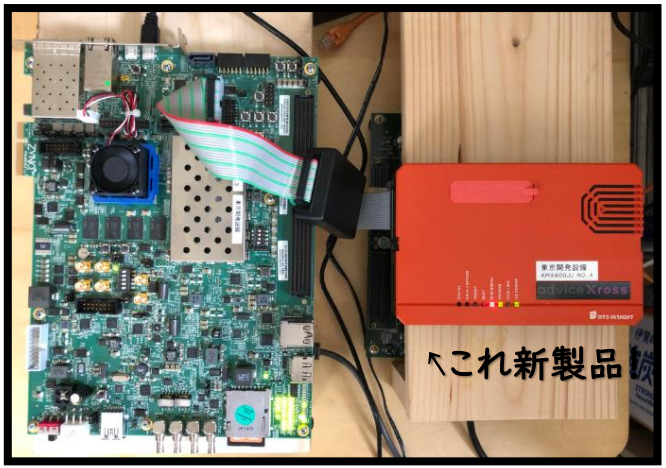
BSP　　：petalinux 2019.1

Debug Target：

　HV　　：Xen-Hypervisor　（4.11.1）

　Dom0　：Backend　Linux　（4.19.0）

　DomU　：Frontend Linux　（4.19.0）



参考：Xenの環境構築

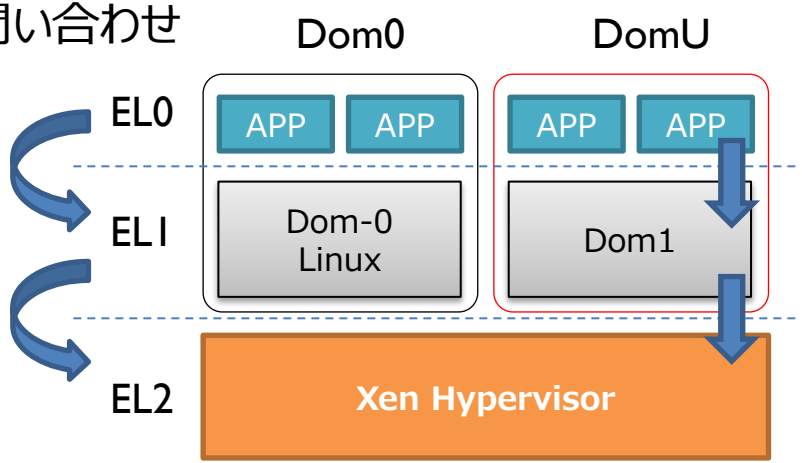
- https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/99188792/Building+Xen+Hypervisor+with+Petalinux+2019.1

Xen on Arm - Debugging

DomUからHypervisorへアクセスするルートをデバッグする

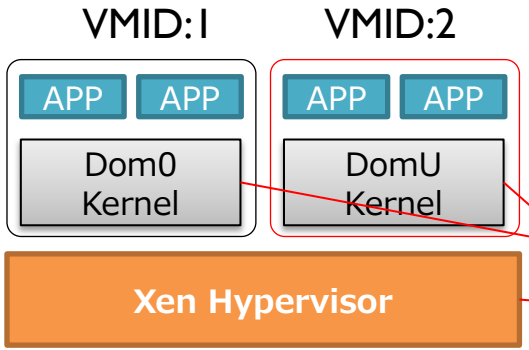
- Dom-UからXenのVersion問合せ
- ～Xen-KernelとDom0とDomUの3つのOSを同時デバッグ

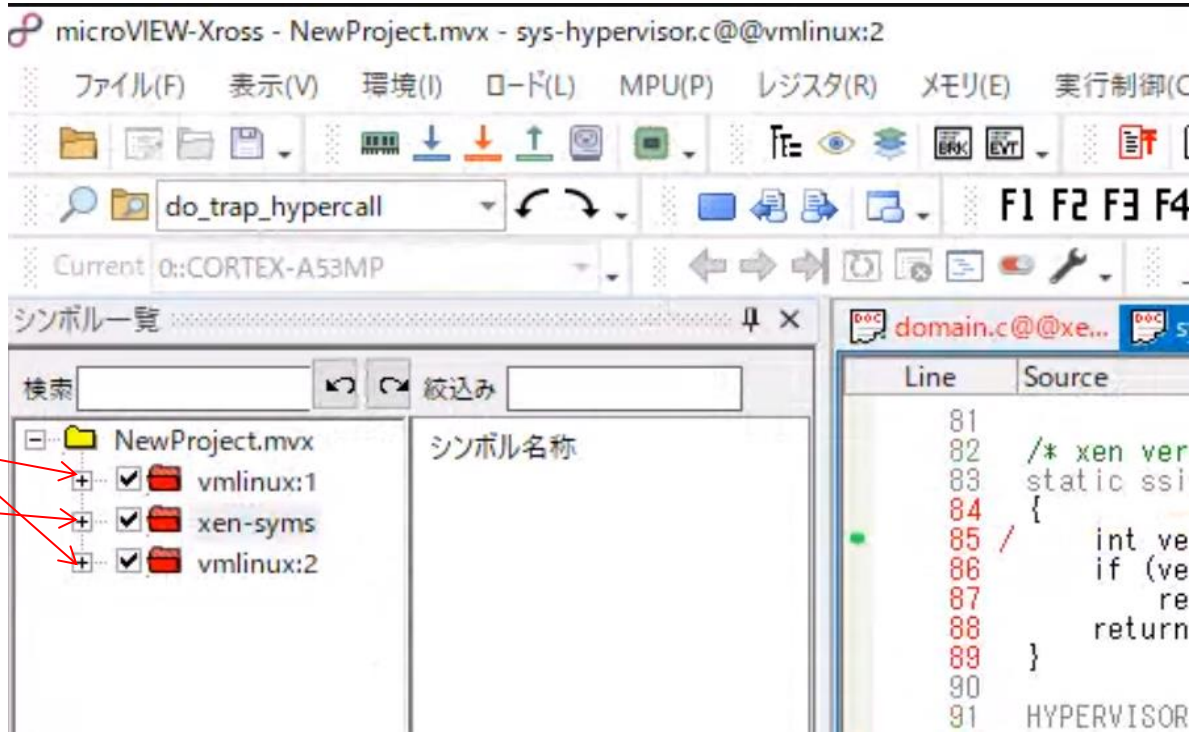
  - ゲストOSにコンソールからXenのVersionを問い合わせ
    - `cat /sys/hypervisor/version/major`
  - ドライバコール　　：EL0→EL1
    - SVC
  - システムコール　　：EL1→EL2
    - HVC
  - 命令トラップ　　：EL2
    - VBAR_EL2　Vectorコード



Xen on Arm – Multi Guest Debug

- VMIDを指定して、各Kernelシンボルを登録





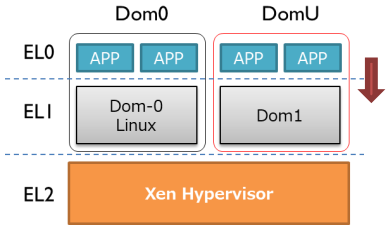
Xen on Arm – System call

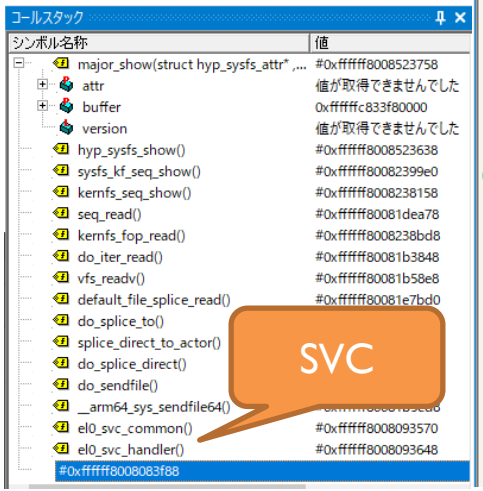
- VM：DomUのコンソールから、Versionをsysfs経由で問合せ

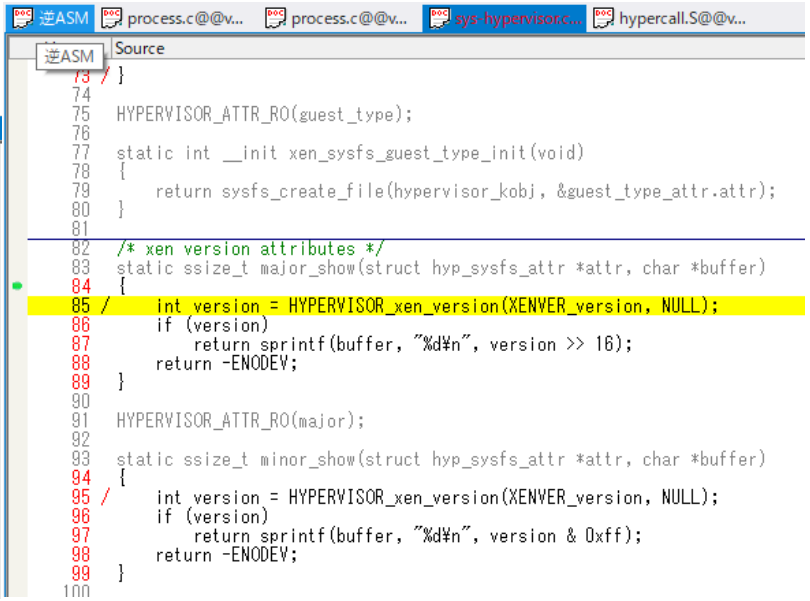
```
cat /sys/hypervisor/version/major
```

drivers/xen/
sys-hypervisor.c

EL0→EL1







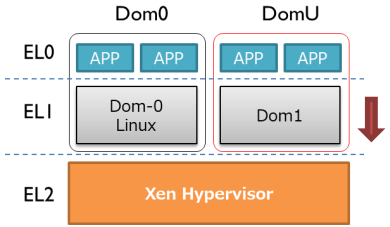
Xen on Arm – System call

- VM：EL1→EL2へ制御を移すためhvc命令を実行

```
cat /sys/hypervisor/version/major
```

drivers/xen/
sys-hypervisor.c

arch/arm64/xen
hypercall.s



sys-hypervisor.c... traps.c@@xen-... splice.c@@vmli... entry.S@@xen-... hypercall.S@@v... sys-hypervisor.c... hypercall.S@@v...

Addr/Line	Data	Source
63		ENDPROC(HYPERVISOR_##hypercall)
64		
65		#define HYPERCALL0 HYPERCALL_SIMPLE
66		#define HYPERCALL1 HYPERCALL_SIMPLE
67		#define HYPERCALL2 HYPERCALL_SIMPLE
68		#define HYPERCALL3 HYPERCALL_SIMPLE
69		#define HYPERCALL4 HYPERCALL_SIMPLE
70		#define HYPERCALL5 HYPE
71		
72		.text
73		
74		HYPERCALL2(xen_version);
FFFFFF800809ECA0	d2800230	HYPERVISOR_xen_v... mov x16,#&0011
FFFFFF800809ECA4	d401d422	hvc #&ea1
FFFFFF800809ECA8	d65f03c0	ret
75		HYPERCALL3(console_io);
FFFFFF800809ECAC	d2800250	HYPERVISOR_conso... mov x16,#&0012
FFFFFF800809ECB0	d401d422	hvc #&ea1
FFFFFF800809ECB4	d65f03c0	ret

HVC

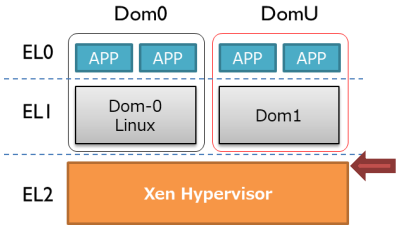
Xen on Arm – vector:entry.S

- Xen：VECTORのアドレスはVBAR_EL2レジスタ

VBAR_EL2

arch/arm/arm64
entry.s

EL2の同期例外に
OCDブレークで罠を貼る



周辺レジスタ

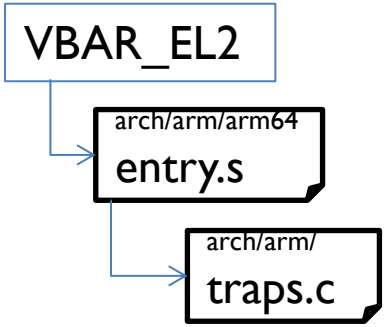
レジスタ名称	値	アドレス
SFR 周辺		
*VFP(A64)		
*ID(A64)		
*MEMORY(A64)		
*SYSCFG(A64)		
*EXCEPTION(A64)		
ESR_EL1	0x56000000	
ESR_EL2	0xffffffff	
ESR_EL3	0xffffffff	
FAR_EL1	0x0000007fb3b847	
FAR_EL2	0xffffff8008005	
FAR_EL3	0x10c010809800	
HPFAR_EL2	0x000000000003	
VBAR_EL1	0xffffff800808	
VBAR_EL2	0x000000000025a000	
VBAR_EL3	0x00000000ffff1000	
AFSR0_EL1	0x00000000	
AFSR0_EL2	0x00000000	
AFSR0_EL3	0x00000000	
AFSR1_EL1	0x00000000	
AFSR1_EL2	0x00000000	
AFSR1_EL3	0x00000000	
IFSR32_EL2	0x00000000	
ISR_EL1	0x00000000	
*SPECIAL(A64)		

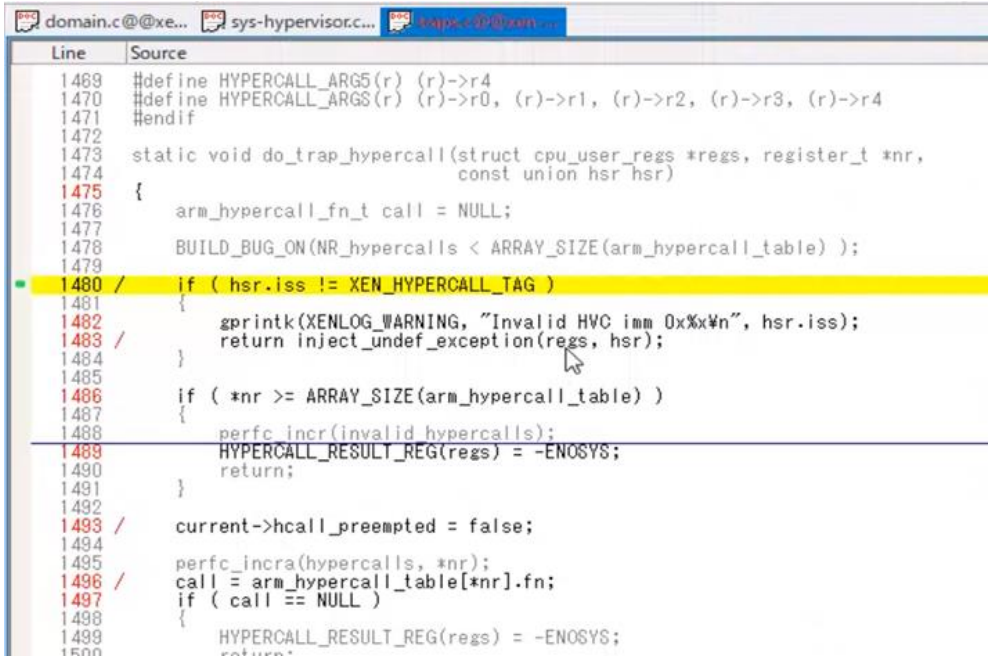
逆ASM | process.c@@v... | process.c@@v... | sys-hypervisor.c... | hypercall.S@@v... | entry.S@@vmli...

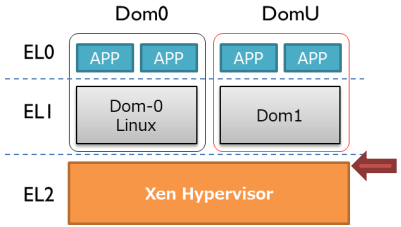
```
Line  Source
 422      .text
 423
 424  /*
 425   * Exception vectors.
 426   */
 427      .pushsection ".entry.text", "ax"
 428
 429      .align  11
 430  ENTRY(vectors)
 431      kernel_ventry   1, sync_invalid         // Synchronous EL1t
 432      kernel_ventry   1, irq_invalid          // IRQ EL1t
 433      kernel_ventry   1, fiq_invalid          // FIQ EL1t
 434      kernel_ventry   1, error_invalid        // Error EL1t
 435
 436      kernel_ventry   1, sync             // Synchronous EL1h
 437      kernel_ventry   1, irq              // IRQ EL1h
 438      kernel_ventry   1, fiq_invalid          // FIQ EL1h
 439      kernel_ventry   1, error            // Error EL1h
 440
 441      kernel_ventry   0, sync             // Synchronous 64-bit EL0
 442      kernel_ventry   0, irq              // IRQ 64-bit EL0
 443      kernel_ventry   0, fiq_invalid          // FIQ 64-bit EL0
 444      kernel_ventry   0, error            // Error 64-bit EL0
 445
 446  #ifdef CONFIG_COMPAT
 447      kernel_ventry   0, sync_compat, 32      // Synchronous 32-bit EL0
 448      kernel_ventry   0, irq_compat, 32       // IRQ 32-bit EL0
 449      kernel_ventry   0, fiq_invalid_compat, 32   // FIQ 32-bit EL0
 450      kernel_ventry   0, error_compat, 32     // Error 32-bit EL0
 451  #else
 452      kernel_ventry   0, sync_invalid, 32     // Synchronous 32-bit EL0
 453      kernel_ventry   0, irq_invalid, 32      // IRQ 32-bit EL0
 454      kernel_ventry   0, fiq_invalid, 32      // FIQ 32-bit EL0
 455      kernel_ventry   0, error_invalid, 32        // Error 32-bit EL0
 456  #endif
 457  END(vectors)
```

Xen on Arm – Traps:EL2

- Xen：VectorからXen-Hypervisorのトラップコード

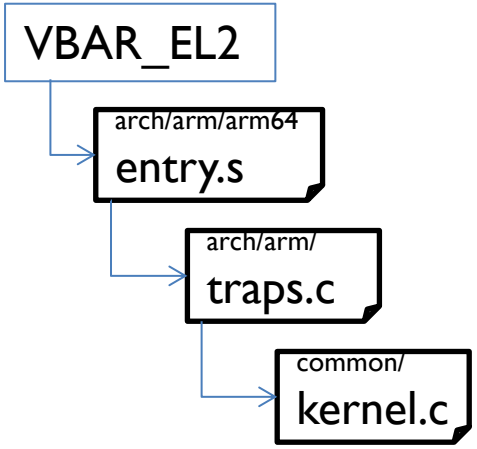


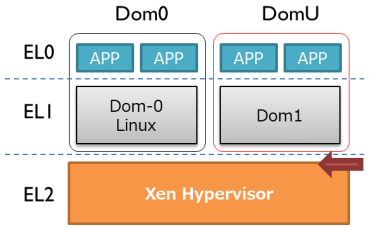




Xen on Arm – Traps:EL2

- Xen：トラップコードから目的のVersion取得処理に到達！





notifier.c@@vm... | kernel.c@@xen... | entry.S@@vmli... | traps.c@@xen-... | rbtree.c@@vmli... | file.c@@vmlinu...

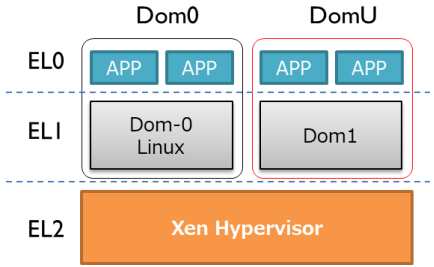
```
Line  Source
328   #endif
329
330   /*
331    * Simple hypercalls.
332    */
333
334   DO(xen_version)(int cmd, XEN_GUEST_HANDLE_PARAM(void) arg)
335   {
336       bool_t deny = !!xsm_xen_version(XSM_OTHER, cmd);
337
338       switch ( cmd )
339       {
340       case XENVER_version:
341           return (xen_major_version() << 16) | xen_minor_version();
342
343       case XENVER_extraversion:
344       {
345           xen_extraversion_t extraversion;
346
347           memset(extraversion, 0, sizeof(extraversion));
348           safe_strcpy(extraversion, deny ? xen_deny() : xen_extra_version());
349           if ( copy_to_guest(arg, extraversion, ARRAY_SIZE(extraversion)) )
350               return -EFAULT;
351           return 0;
352       }
353
```

仮想環境の可視化

仮想環境で何を可視化したいか？

目的はなにか？

- 例えば・・・パフォーマンス観点
  - 自分の処理は遅くないよね・・・？
  - バックエンドが遅いんじゃないかな？
- もしくは・・・なぜ自分のOSに処理がまわってこないのか？
  - 自分のOSは設計通りスケジューリングされているか？
  - vCPUのスケジューリングに問題があるのでは？



大きく2つに分けて考える（とりあえず）

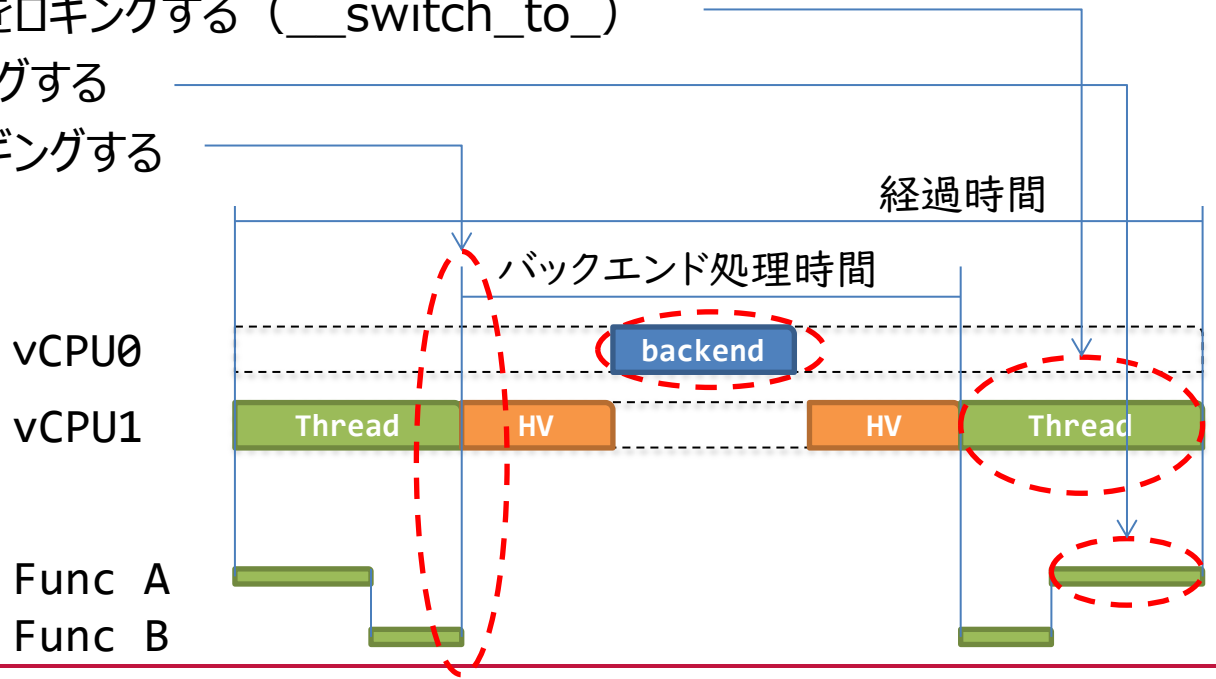
- 前者は、GuestOS視点
  - 自分のアプリがダメなのか、バックエンドの時間なのか、切り分けをしたい
- 後者は、システム視点
  - vCPUのスケジューリングとその根拠、HVの仕事の分析をしたい。

プログラムの実行時間を測定するには？

- VMのソースに埋め込む：今までと同じ

  APIライブラリ等によるhook関数の埋め込み

  – context switchをロギングする（__switch_to_）
  – 関数コールをロギングする
  – システムコールをロギングする



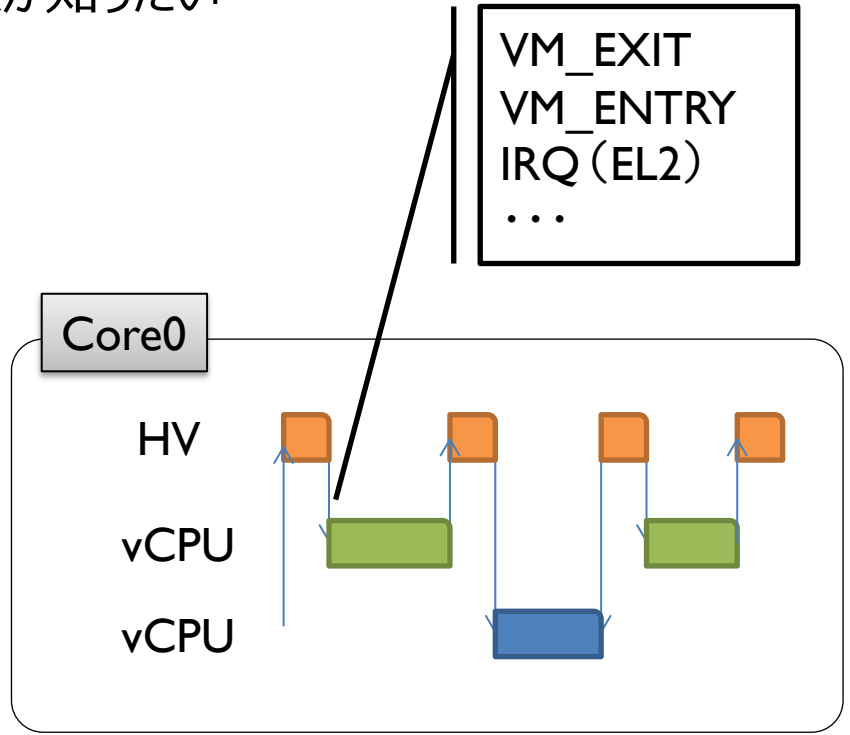
HVシステム全体を見れないか？

- 自分のVMだけでなく全体のロードバランスが知りたい
  - HVのvCPUスケジューリングは？
- HVの実行時間が知りたい
  - HVの介入による時間はどの程度なのか？

しかし

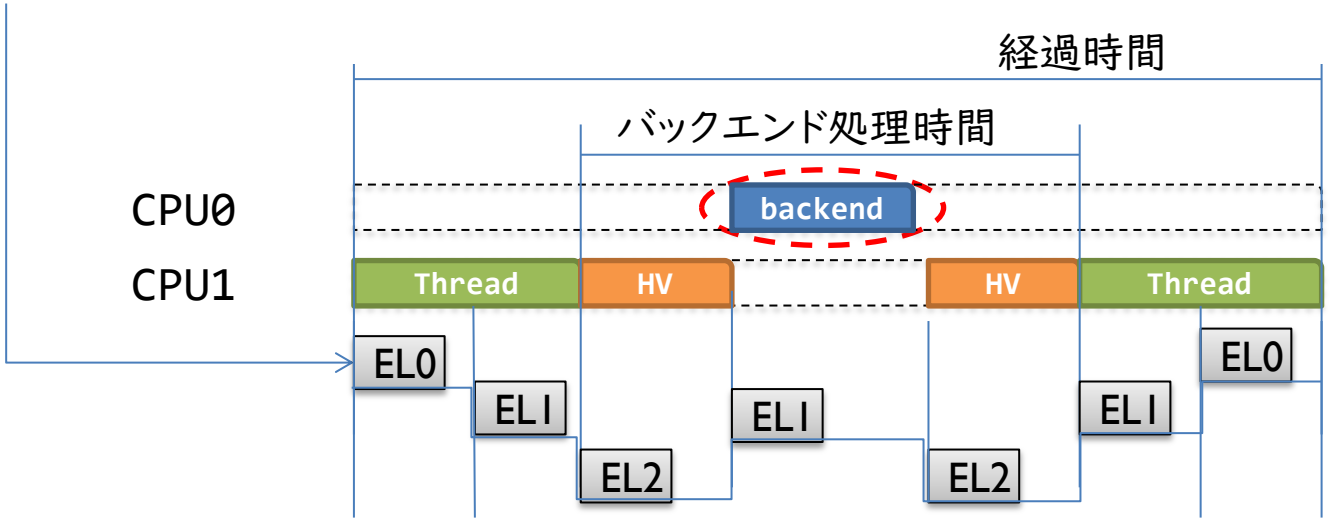
HVに埋め込むことは出来ない

- OSSではない
- デバッグ用hookポイントがあるかも？



HVシステムの実行時間を測定するには？

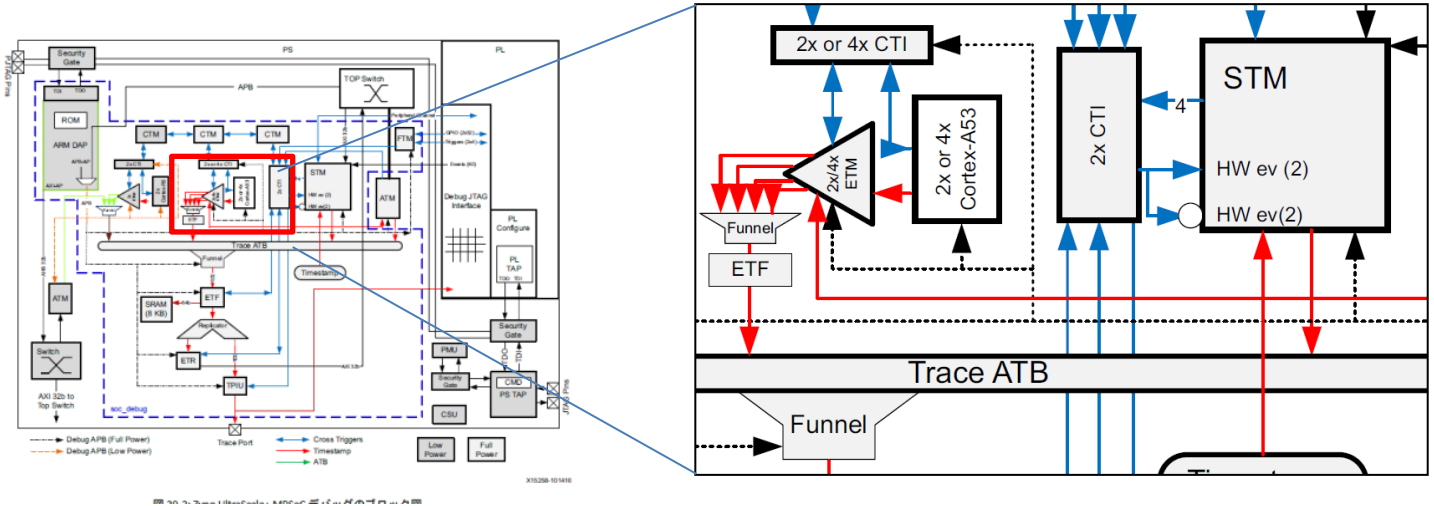
- HVの外側から観測するには？
  - CPUのイベントをデバッグIFを使用してロギングする
  - Arm® CoreSight™　ETM
    - **Virtual context identifier tracing :** 仮想コンテキスト識別子トレース



CoreSightでシステムトレースに有効なものは？

- Arm® CoreSight™ コンポーネント
  - ETM : Virtual context identifier tracing（EL2）
  - STM: ソフトウェアトレースをHWで収集

Zynq UltraScale+ MPSoC のデバッグシステム



https://japan.xilinx.com/support/documentation/user_guides/j_ug1085-zynq-ultrascale-trm.pdf[　出典：UG1085 （v1.4）

CoreSight：STMを使うには・・・

- Linux：STM（System Trace Module）
  - Linaroの記事にあります。
  - Arm純正のリファレンスボードの例があります。

```
stm@10006000 {
            compatible = "arm,coresight-stm", "arm,primecell";
            reg = <0 0x10006000 0 0x1000>,
                  <0 0x01000000 0 0x1800000>;
            reg-names = "stm-base", "stm-stimulus-base";
            clocks = <&clk26mhz>;
            clock-names = "apb_pclk";
            port {
                        stm_out: endpoint {
                                    remote-endpoint = <&funnel_in_port4>;
                        };
             };
};
```

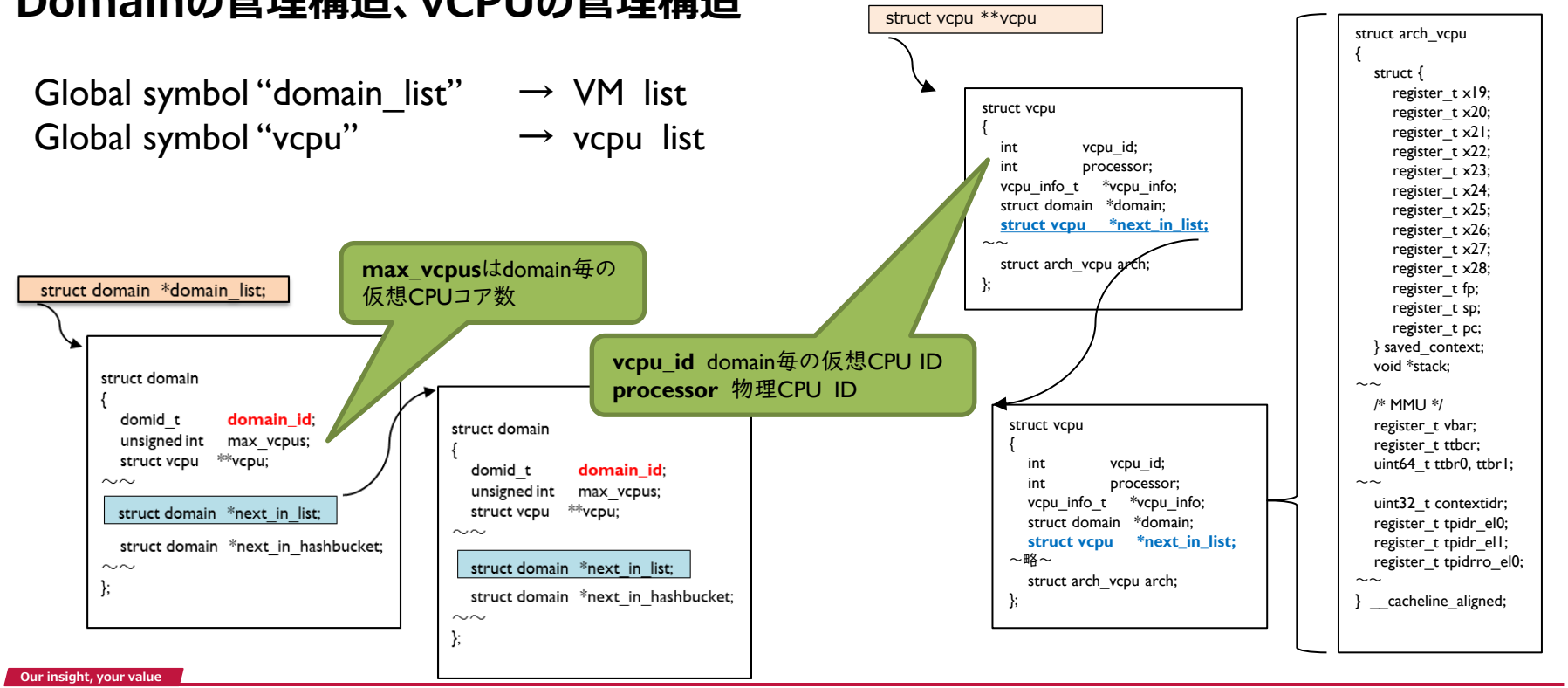
- 残念ながら、まだ簡単には使えないようです。
  - XilinxのSoCに、STMは搭載されています。
  - TRMにも仕様の記載があります。
  - しかし、Device TreeにCoreSightの記載がありません・・・
  - menuconfigで有効にするだけでは使えません。

このへんは、またの機会に

OSSだから見える事：Xen on Arm

Domainの管理構造、vCPUの管理構造

Global symbol “domain_list” → VM list
Global symbol “vcpu” → vcpu list



OSSだから見える事：Xen on Arm

vCPUのスケジューリング

arch/arm/domain.c：context_switch（ ）
arch/arm/arm64/entry.S：__context_switch

ここにAPIライブラリを挿入して
vcpuのスケジューリングを可視化

domain.c (~/petalinux/work/xilinx-zcu106-2019.1/build/t...4.11+gitAUTOINC+e4547c6

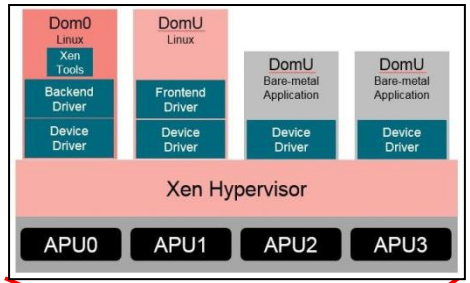
Open Save

```
void context_switch(struct vcpu *prev, struct vcpu *next)
{
    ASSERT(local_irq_is_enabled());
    ASSERT(prev != next);
    ASSERT(!vcpu_cpu_dirty(next));

    if ( prev != next )
        update_runstate_area(prev);

    local_irq_disable();

    /*
     * If the serrors_op is "FORWARD", we have to prevent forwarding
     * SError to wrong vCPU. So before context switch, we have to use
     * the SYNCRONIZE_SERROR to guarantee that the pending SError would
     * be caught by current vCPU.
     *
     * The SKIP_CTXT_SWITCH_SERROR_SYNC will be set to cpu_hwcaps when the
     * serrors_op is NOT "FORWARD".
     */
    SYNCHRONIZE_SERROR(SKIP_CTXT_SWITCH_SERROR_SYNC);

    set_current(next);

    prev = __context_switch(prev, next);

    schedule_tail(prev);
}
```

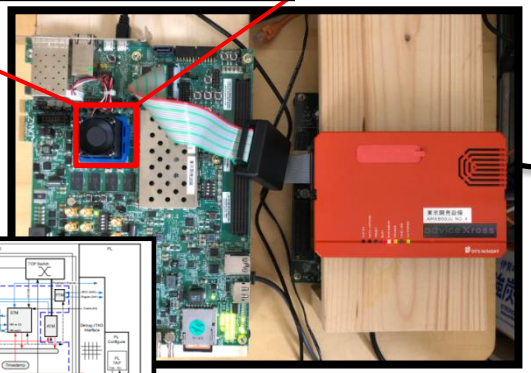
C Tab Width: 8 Ln 353, Col 36 INS

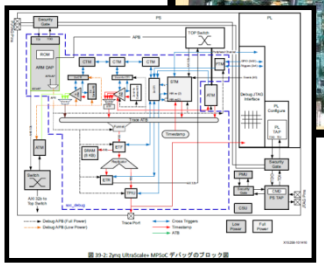
まとめ

- HVのデバッグ
  - **ゲストOSの同時デバッグ**は、AArch64 Virtualizationの情報で**実現できた！**
- 可視化：VM
  - HVやvCPUの処理は見えないが、EL1の動作でバックエンドの時間を推測可能
- 可視化：外から観測
  - **CoreSight ETM**を活用すれば、contextの実行履歴が取得できる。
    - ただし、使うためのハードルが高い
- 可視化：HVの中に期待
  - Hypervisorが直接トラップする例外やスケジューラは見えない。
  - vCPUのスケジューリングなどは、HVの**トレースポイントの公開**を**期待！！**



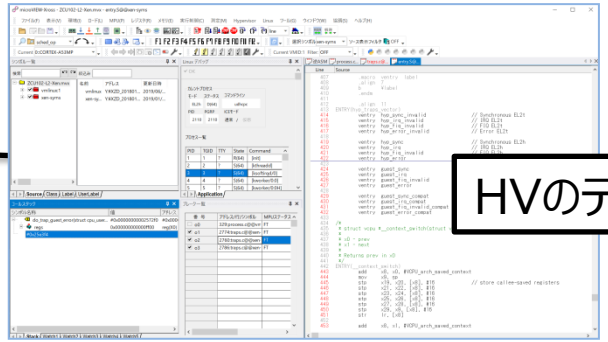
Xen on Arm





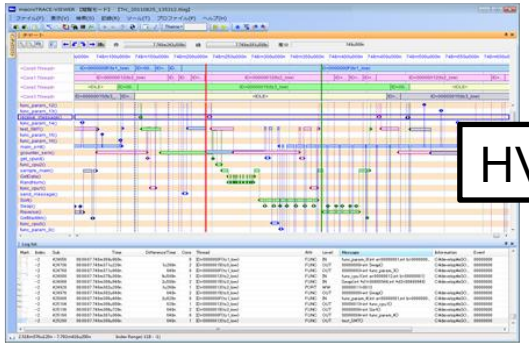
adviceXross

Arm® CoreSight™ ETM/STM



HVのデバッグ

microVIEW-Xross



HVの可視化

macroTRACE-VIEWER

本日はご視聴頂きましてありがとうございました。

本日の内容でもっと知りたいと言う方はご連絡をください！

info-advice@dts-insight.co.jp

是非、最適な環境で開発を行っていきましょう！

Our insight, your value

Resource

- https://static.docs.arm.com/100942/0100/aarch64_virtualization_100942_0100_en.pdf
- https://wiki.xenproject.org/wiki/Category:XenARM
- https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842530/XEN+Hypervisor
- https://static.docs.arm.com/100942/0100/aarch64_virtualization_100942_0100_en.pdf
- https://www.linaro.org/blog/stm-and-its-usage/