動的バイナリ変換による RISC-V の M-mode 仮想化の高速化とジッタの低減

鬼頭 太郎[1,a)]　山本 遼介[2,b)]　増田 大樹[2,c)]　毛利 公一[1,d)]

概要：組込みシステムでは，リアルタイム OS（RTOS）と汎用 OS を単一のハードウェア上で実行する要求がある．それに対し，RISC-V 向けハイパーバイザ Violet は，M-mode で動作する RTOS を仮想マシン (VM) 上で実行可能にしている．ただし，M-mode 用 Control and status register (CSR) アクセスのたびにハイパーバイザによるトラップとエミュレートを行う．その結果，RTOS の割込み処理やタスク切り替えの実行時間が増大し，ジッタが生じる．そこで，本論文では，命令単位の動的バイナリ変換によって M-mode エミュレーションの高速化とジッタを低減する方式を提案する．初回トラップ時に命令を解析し，VM 内で実行可能な命令へ直接置換するか，生成したエミュレーションコードへジャンプするように書き換える．これにより，2 回目以降にハイパーバイザへのトラップを不要にする．生成するコードには条件分岐とループを用いず，キャッシュカラーリングと組み合わせてジッタを抑える．本方式により，定常時の CSR アクセスの最大実行時間を，直接置換では従来方式の約 130 分の 1，生成したコードの実行では約 11–15 分の 1 に短縮し，ジッタを 4 サイクル以下に抑えた．また，FreeRTOS の協調スケジューリングにおけるスループットを従来方式の約 11.6 倍に向上させた．

キーワード：RISC-V，仮想化，ハイパーバイザ，動的バイナリ変換，リアルタイムシステム

Accelerating RISC-V M-mode Virtualization and Reducing Jitter with Dynamic Binary Translation

1. はじめに

組込みシステムでは，制御処理を担うリアルタイム OS（RTOS）と，ユーザインタフェースやネットワーク処理を担う汎用 OS を，単一のハードウェア上で実行する要求がある．こうした要求に対し，組込み向けハイパーバイザによる複数 OS の実行環境の統合が研究されている [1], [2], [3], [4]．複数の OS を共存させるには，それぞれの実行環境を分離するとともに，仮想化に伴う実行時間の増大と変動（ジッタ）を抑える必要がある．

RISC-V の Hypervisor 拡張は S-mode と U-mode の仮想化を支援するが，最上位の特権レベルである M-mode は仮想化の対象としていない [6]．一方，RISC-V 向け RTOS には M-mode で動作するものがあり，そのままでは Hypervisor 拡張を用いた仮想マシン（VM）上で実行できない．そこで，組込み向けハイパーバイザ Violet は，M-mode の Control and Status Register（CSR），特権命令，割込み・例外処理をソフトウェアでエミュレートし，ゲストに M-mode の実行環境を提供する．この M-mode 仮想化により，M-mode 向け RTOS を無改変のまま VS-mode 上で実行する完全仮想化を実現する [5], [7], [8]．しかし，従来のトラップ&エミュレート方式では，CSR アクセス命令を実行するたびにハイパーバイザへの遷移と命令解析が発生する．割込みハンドラやコンテキストスイッチで CSR アクセスが繰り返されると，このオーバヘッドが RTOS の性能を低下させる．

本論文では，M-mode 向け RTOS の完全仮想化を維持し

[1] 立命館大学
Ritsumeikan University
[2] 三菱電機株式会社 設計技術開発センター
Engineering and Development Center, Mitsubishi Electric Corporation
[a)] tkito@asl.cs.ritsumei.ac.jp
[b)] Yamamoto.Ryosuke@ce.MitsubishiElectric.co.jp
[c)] Masuda.Hiroki@dc.mitsubishielectric.co.jp
[d)] mouri@asl.cs.ritsumei.ac.jp

ⓒ2026 Information Processing Society of Japan

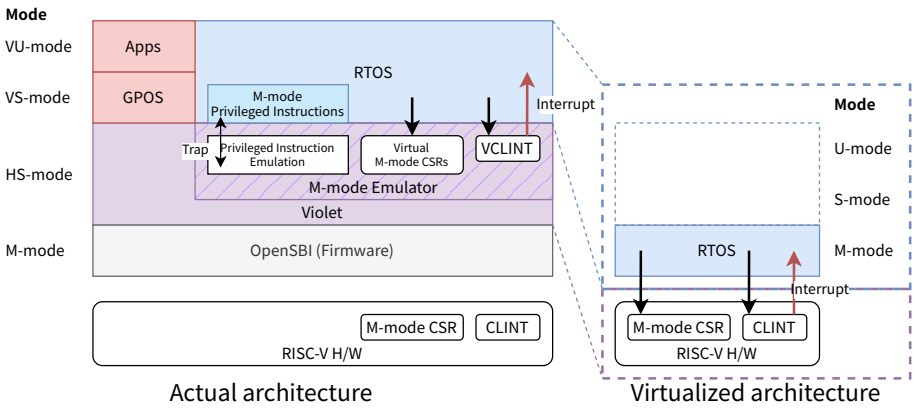

図 1　Violet の構成と M-mode エミュレーション [5]

つつ，M-mode 仮想化に伴うエミュレーションを高速化し，ジッタを抑制する動的バイナリ変換方式を提案する．提案方式は，エミュレーション対象の命令単位で変換を行い，元の命令を置換する．対象命令の初回実行時に元命令を VS-mode で実行可能な命令，またはエミュレーションコードへのジャンプ命令に置換する．2 回目以降は VM 内でエミュレーションを完結させ，CSR アクセスに伴うハイパーバイザへの遷移を削減する．さらに，生成コード，シャドウ CSR などのデータ，およびページテーブルにキャッシュカラーリングを適用し，実行時間の変動を抑制する．

本論文の貢献は，次の 3 点である．

(1) 命令の直接置換と命令位置ごとのコード生成に，トラップ時の状態更新と例外復帰の処理を組み合わせた M-mode エミュレーション方式を示した．
(2) シャドウ CSR を用いた割込み状態の管理，条件分岐とループを用いない CSR エミュレーション，コード・データ・ページテーブルのキャッシュカラーリングを組み合わせ，実行時間の変動を抑制した．
(3) 実機上で CSR アクセス時間とその分布，RTOS のスループット，初回実行時間を評価し，提案方式の効果と制約を明らかにした．

2 章では背景と従来方式の課題を述べる．3 章で提案方式を示し，4 章で実装を説明する．5 章で評価結果を示し，6 章で適用範囲と制約を議論する．7 章で関連研究を述べ，8 章でまとめる．

2. 背景と従来方式の課題

2.1 RISC-V の特権機構と仮想化支援機能

RISC-V には，最上位の M-mode，OS 向けの S-mode，アプリケーション向けの U-mode という特権レベルがある [6]．

割込みと例外によるハンドラへの遷移をトラップと呼ぶ．M-mode では，トラップの要因を `mcause` に記録し，`mtvec` の設定に従ってハンドラへ制御を移す．トラップからの復帰には，M-mode では `MRET` 命令，S-mode では `SRET` 命令を用い，CSR に保存された復帰先へ制御を移すとともに，割込み許可状態を復元する [6]．

Hypervisor 拡張は，ハイパーバイザを実行する HS-mode と，ゲスト OS およびゲストアプリケーションを実行する VS-mode，VU-mode を提供する．VS-mode で S-mode の CSR アクセス命令を実行すると，対応する VS-mode の CSR へアクセスする．例えば，VS-mode での `sstatus` へのアクセスは `vsstatus` へのアクセスとして実行される．これにより，S-mode で動作する Linux などの汎用 OS を，無変更で VM 上で実行できる [6]．

一方，M-mode の CSR へのアクセスや `MRET` 命令は VS-mode では直接実行できず，エミュレーションを必要とする [6]．

2.2 Violet の概要

図 1 に RISC-V 向け組込みハイパーバイザ Violet と，Violet の機能である M-mode Emulator の構成を示す．Violet は HS-mode で動作し，S-mode 向けの汎用 OS は Hypervisor 拡張によって VS-mode で実行する．M-mode 向けの RTOS も，Violet の M-mode Emulator を介して VS-mode で実行する．M-mode Emulator は，M-mode の CSR アクセスと特権命令をエミュレートする．また，M-mode と VS-mode の間で割込み・例外の種別を変換する．さらに，タイマ制御に用いるデバイスである Core-Local Interruptor (CLINT) へのアクセスを仮想化する [5]．M-mode と VS-mode では，CSR のビット配置や割込み・例外に対応する例外コードが異なる．例えば，M-mode の割込み許可ビット `mstatus.MIE` に対応する動作を，VS-mode では `vsstatus.SIE` によって実現する．したがって，M-mode の CSR 番号を VS-mode の CSR 番号へ置き換えるだけでは，すべての CSR 操作をエミュレートできない．ゲストから見える CSR の値と，ハードウェアが使用する状態を対応付ける処理が必要となる．

図 2〜4 に，Violet における主要なビットフィールドの対応を示す [5]．CSR への書き込みではゲストが指定した

ⓒ2026 Information Processing Society of Japan

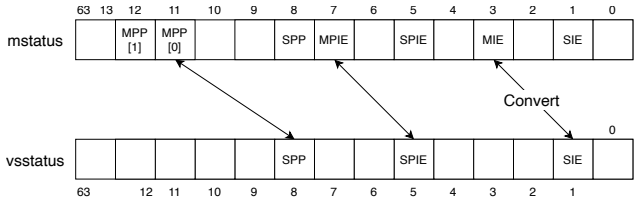

図 2 `mstatus` と `vsstatus` のビット対応 [5]

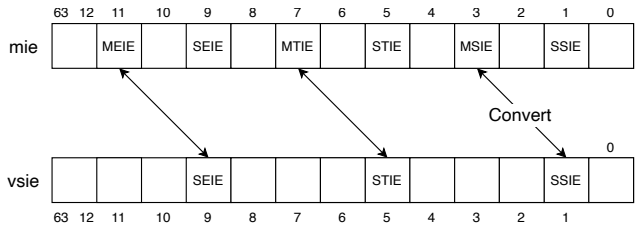

図 3 `mie` と `vsie` のビット対応 [5]

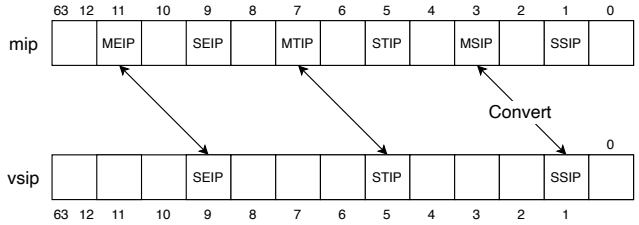

図 4 `mip` と `vsip` のビット対応 [5]

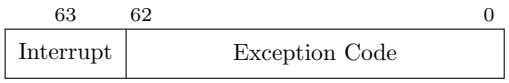

図 5 `mcause` と `vscause` のビット構成（RV64）

M-mode の値を VS-mode の配置へ変換し，読み出しでは逆方向に変換する．`mie` と `mip` では，ソフトウェア・タイマ・外部割込みに対応する 3 ビットをそれぞれ移動する．

図 5 に，RV64 における `mcause` と `vscause` のビット構成を示す．最上位の Interrupt ビットが 1 なら割込み，0 なら例外を表し，残りの Exception Code フィールドは割込み・例外の種別を示す [6]．

表 1 に，M-mode と VS-mode の割込み・例外要因と，`mcause` および `vscause` の値の対応を示す．$I = 2^{63}$ は，割込みを示す Interrupt ビットの値である．例えば，タイマ割込み発生時に `mcause` を読み出す場合，`vscause` から $I+5$ を読み出し，Violet が $I+7$ へ変換して返すことで，ゲストは M-mode のタイマ割込みが発生したと認識する．また，VS-mode で `ECALL` 命令を実行すると，Exception Code フィールドの値が 10 の環境呼出し例外が HS-mode へ通知される [6]．Violet はこの例外をゲストへ配送し，`mcause` の読み出し時に M-mode からの環境呼出しを表す 11 へ変換する．

2.3 トラップ&エミュレート方式の課題

従来方式では，ゲストによる M-mode の CSR アクセス命令を実行した際に，不正命令例外が発生し，Violet へ制御が移る．Violet はゲストのレジスタを退避し，例外が発生した位置の命令を取得する．続いて，CSR 番号，アクセス種別，ソース・宛先レジスタを解析し，仮想 CSR または対応する VS-mode の CSR に対して操作を行う．読み出した値をゲストのレジスタへ反映した後，プログラムカウンタ（PC）を次の命令へ進め，ゲストの実行を再開する．

同じ命令を繰り返し実行する場合も，Violet への遷移，レジスタの退避・復元，命令解析が毎回発生する．RTOS の割込み処理やコンテキストスイッチでは，割込みの禁止・許可，復帰先の設定のために，CSR アクセス命令が繰り返し実行される．その結果，オーバヘッドが処理全体の実行時間を増大させる．従来方式の評価でも，CSR アクセス時間の増大と変動，および FreeRTOS の性能低下が報告されている [5]．そこで本論文では，繰り返し実行される CSR アクセス命令に着目し，トラップの削減とエミュレーション処理の実行時間変動の抑制を図る．測定条件と結果は 5 章に示す．

3. 動的バイナリ変換による M-mode エミュレーション

提案方式は，対象命令の処理を VM 内で実行できる形に変換し，繰り返し実行時のハイパーバイザへの遷移を削減する．さらに，エミュレーションの実行経路とキャッシュへの干渉を制御し，実行時間の変動を抑える．

3.1 初回実行時の命令変換

図 6 に，元命令をエミュレーションコードへのジャンプ命令に置換する場合の実行経路を示す．初回実行時には従来方式と同様にトラップが発生し，Violet が対象命令を解析する．その処理を VM 内で実行可能な 1 命令で代替できる場合は，元命令を直接置換する．複数命令による処理が必要な場合は，エミュレーションコードを生成して VM 内に配置し，元命令をそのコードへのジャンプ命令に置換する．

3.2 命令置換後の実行

命令置換後は元の命令位置から実行を再開し，置換命令またはエミュレーションコードをゲストが直接実行する．以後，同じ命令位置に到達した際にも，適用済みの変換結果を利用するため，ハイパーバイザへの遷移や命令解析を繰り返す必要がない．

3.3 実行時間変動の抑制

トラップを削減しても，実行経路やキャッシュの影響による実行時間の変動は残る．CSR エミュレーションでは，

ⓒ2026 Information Processing Society of Japan

表 1　割込み・例外要因と CSR 値の対応 [5], [6]

M-mode の割込み・例外要因	`mcause`	VS-mode の割込み・例外要因	`vscause`
Machine software interrupt	$I+3$	Supervisor software interrupt	$I+1$
Machine timer interrupt	$I+7$	Supervisor timer interrupt	$I+5$
Machine external interrupt	$I+11$	Supervisor external interrupt	$I+9$
Environment call from M-mode	11	Environment call from VS-mode	10

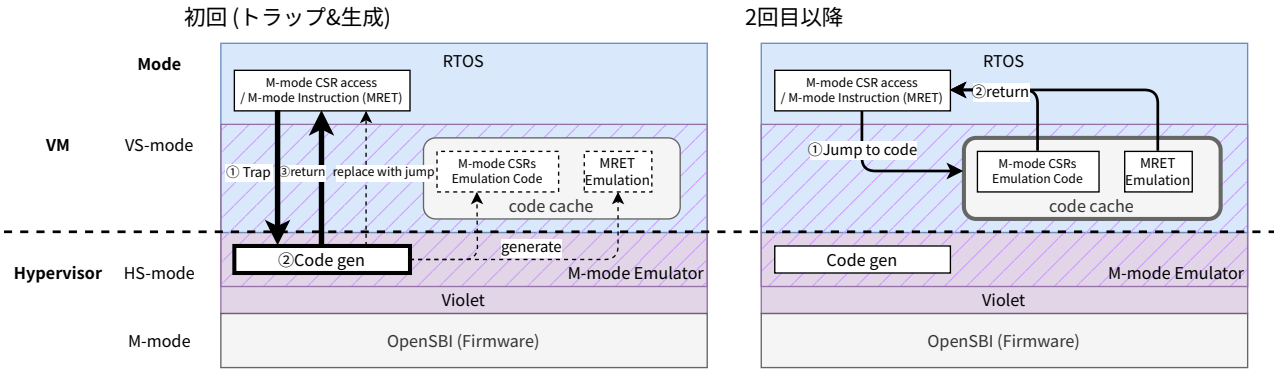

図 6　初回実行時と命令置換後の処理経路

入力値によって実行する命令列が変わると，実行命令数や分岐予測の成否によって処理時間が変動する．提案方式では，生成コードに条件分岐と実行時のループを用いず，同じ対象命令に対して一定の命令列を実行することで，この変動を抑える．

また，エミュレーションに必要なコードやデータがゲスト OS のアクセスによってキャッシュから追い出されると，再取得による遅延が発生する．そこで，キャッシュカラーリングによって両者が使用するキャッシュ領域を分離する．エミュレーション用のコード・データに加え，それらのアドレス変換に必要なページテーブルも分離の対象とし，ゲスト OS のアクセスによる実行時間への影響を抑える．

4. 実装

本章では，命令解析とコード生成の概要を示し，専用データ領域を用いたエミュレーションの実装を述べる．その上で，生成コードの配置と命令の置換，キャッシュ競合を抑えるカラーリングの適用方法を説明する．

4.1 命令解析とコード生成

不正命令例外を起こした CSR アクセス命令から，対象 CSR，操作種別，ソース・宛先レジスタを特定する [9]．表 2 に，処理対象ごとの実装方式を示す．直接置換の対象である `mtval` は，元命令の CSR 番号を `stval` へ変更し，操作種別とソース・宛先レジスタを保持する．

複数命令を必要とする CSR 操作には，Just-In-Time（JIT）コンパイルによって命令位置ごとにコードを生成する．CSR 番号，操作種別，オペランドレジスタ，元命令の次の PC を生成コードへ埋め込み，実行時の命令解析を省略する．CSR とソースレジスタの値は実行時に取得するため，入力値が変化しても同じコードを利用できる．同じ命令が異なる位置に存在する場合は，復帰先が異なるため個別にコードを生成する．

MRET エミュレーションでは復帰先をシャドウ `mepc` から取得するため，コードを命令位置ごとに生成する必要がなく，同一 VM 内で共有する．このコードと，トラップ発生時の状態更新を行うコードは VM 初期化時に生成する．`MRET` 命令の初回実行時には，生成済みのコードへのジャンプを適用する．

4.2 エミュレーション専用データ領域

生成したエミュレーションコードはゲストと同じ VS-mode で動作し，ゲストのレジスタを作業用に使用する．そのため，仮想 CPU（VCPU）ごとにレジスタ退避用の専用スタックを用意する．また，ゲストから見える CSR の値をメモリ上で保持するシャドウ CSR も設ける．

図 7 に，専用スタックとシャドウ CSR をまとめたデータ領域の配置を示す．4 VCPU 分の領域を 1 枚の 4 KiB ページに格納する．

図中の矢印は，前処理を終えた時点で各レジスタが指す位置を示す．`sp` は専用スタックを指し，その位置はレジスタの退避量に応じて変わる．`s0` はシャドウ CSR の先頭アドレスを保持し，CSR の読み書きや，ゲストへの復帰時に割込み許可状態を取得するために用いる．

4.3 CSR エミュレーションの実装

CSR エミュレーションは，レジスタを退避する前処理，CSR 操作を行う本体，実行状態を復元する後処理で実装す

ⓒ2026 Information Processing Society of Japan

表 2　CSR・特権命令と処理方式の対応

処理方式	対象	主な処理
直接置換	`mtval`	CSR 番号を `stval` へ変更
個別生成	`mepc`, `mscratch`	シャドウ CSR への読み出し・書き込み
	`mstatus`	シャドウ CSR とハードウェアの割込み状態を同期
	`mtvec`	シャドウ CSR を更新し，トラップエントリを選択
	`misa`, `mvendorid`, `marchid`, `mimpid`	定数を返す
	`mideleg`, `medeleg`	ゲスト S-mode へ委譲しない構成として 0 を返す
	`mhartid`	仮想 CPU の識別子を返す
	`mie`, `mip`	割込みビットの位置を変換
	`mcause`	Exception Code フィールドの値を変換
VM 初期化時に生成	`MRET`	シャドウ CSR に基づく例外復帰
	トラップエントリ	シャドウ CSR の更新とゲストハンドラへの分岐

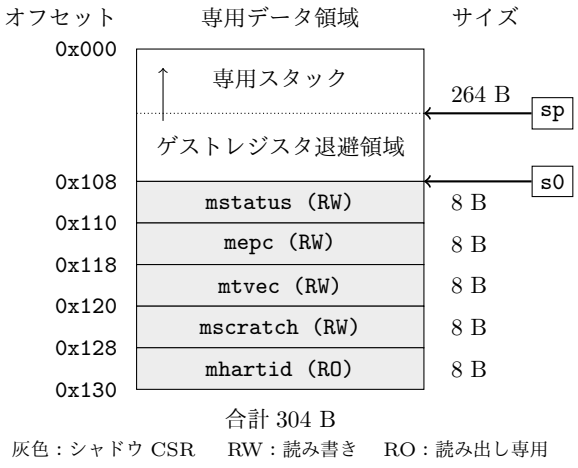

図 7　専用データ領域のレイアウト

る．レジスタの退避と CSR の状態保持には，4.2 節の専用領域を用いる．処理途中にゲストの割込みハンドラへ制御が移ることを防ぐため，前処理の先頭で割込みを禁止し，実行状態の復元後にゲストの割込み許可状態を反映する．対象 CSR ごとの命令列の生成は 4.4 節で述べる．

擬似コード 1 に，CSR エミュレーションの実行手順を示す．S はシャドウ CSR，pc_{next} は元命令の次の PC，Memory[a] はアドレス a の内容を表す．生成コードは VS-mode で動作し，S-mode CSR へのアクセスを通じて，対応する VS-mode の CSR を操作する（2.1 節）．

前処理では，まず `sstatus.SIE` を 0 にして割込みを禁止し，`sp` と `sscratch` を交換する．`sscratch` には，あらかじめ専用スタック内の `s0` 退避領域のアドレスを設定しておく．この交換により，ゲストの `sp` を `sscratch` に保持したまま，専用スタックへ切り替える．次に，ゲストの `s0` を退避し，その直後に配置したシャドウ CSR の先頭アドレスを `s0` に設定する．SaveRegisters では，残りの一時レジスタの退避と，退避量に応じた `sp` の更新を行う．退避するレジスタは対象命令に応じてコード生成時に決定する．退避・復元の命令列も生成時に展開し，実行時のループを用いない．

擬似コード 1 CSR エミュレーションの実行

```
1: procedure EmulateCsr
2:     sstatus.SIE ← 0
3:     Swap(sp, sscratch)
4:     Memory[sp] ← s0
5:     s0 ← sp + 8
6:     SaveRegisters(sp)
7:     CsrBody
8:     ResumeGuest(pc_next, S.mstatus.MIE)
9: end procedure

10: procedure ResumeGuest(pc, ie)
11:     sepc ← pc
12:     sstatus.SPIE ← ie
13:     sstatus.SPP ← 1
14:     RestoreRegisters(sp)
15:     s0 ← Memory[sp]
16:     Swap(sp, sscratch)
17:     SRET
18: end procedure
```

続いて，CsrBody で対象 CSR の読み書きや値変換を行う．CSR の旧値を返す命令では，元命令の宛先レジスタへ結果を格納する．この部分の命令列は，4.4 節で述べる方法で生成する．

後処理の ResumeGuest には，復帰先の PC と，シャドウ `mstatus.MIE` に保持したゲストの割込み許可状態を渡す．復帰先を `sepc` に，割込み許可状態を `sstatus.SPIE` に設定し，VS-mode へ復帰するため `sstatus.SPP` を 1 にする．続いて，RestoreRegisters で一時レジスタと `sp` を復元する．最後に `s0` を復元し，`sp` と `sscratch` の交換によってゲストのスタックへ戻す．

ⓒ2026 Information Processing Society of Japan

```
# Original instruction at P
csrrw a1, mepc, a0

# Replacement at P
jal   x0, csr_emulation_code

# CSR operation in csr_emulation_code
# s0: base address of shadow CSRs
addi  t6, s0, 8
ld    a1, 0(t6)
sd    a0, 0(t6)
```

図 8　csrrw a1, mepc, a0 の変換例

最後に SRET を実行すると，sepc の指す命令から実行を再開すると同時に，sstatus.SPIE の値が sstatus.SIE へ反映される．これにより，レジスタとスタックの復元を終えてから，ゲストの割込み許可状態を反映する．

4.4　CSR 操作のコード生成

個別生成の処理本体は，シャドウ CSR へのアクセス，定数の読み出し，実 CSR との値変換に分類できる．対象 CSR に必要な処理を，元命令の操作種別とオペランドに合わせて生成する．mstatus では，シャドウ CSR による状態保持と実 CSR との値変換を組み合わせる．以下では，各処理の生成方法を順に述べる．

4.4.1　シャドウ CSR へのアクセス

シャドウ CSR への読み出し・書き込みは，対応するメモリ領域へのロード・ストアに変換する．ビットのセット・クリアを行う CSR アクセス命令では，読み出した値へ論理和やマスク演算を適用してから書き戻す．CSR の旧値を返す命令では，更新前の値を宛先レジスタへ格納する．mepc と mscratch は，これらのメモリ操作によってエミュレートする．

mstatus では，シャドウ CSR で管理する割込み許可状態を，4.3 節の後処理でハードウェアへ反映する．mtvec の書き込みでは，シャドウ CSR の更新に加え，設定されたモードに応じてトラップエントリを選択する．この選択とトラップ発生時の処理は 4.5 節で述べる．

図 8 に，csrrw a1, mepc, a0 の変換例を示す．この命令は，mepc の旧値を a1 へ読み出し，a0 の値で mepc を更新する．元命令は CSR エミュレーション用のコードへのジャンプに置換される．図中の s0 には，前処理でシャドウ CSR 領域の先頭アドレスを設定しておく．生成コードの本体は，シャドウ mepc のアドレスを求め，旧値を a1 へ読み出し，a0 の値を格納する 3 命令となる．この 3 命令が擬似コード 1 の CsrBody に相当する．前処理と後処理によってゲストの実行状態を管理し，元命令の 4 バイト後から実行を再開する．

4.4.2　定数の読み出し

misa，mvendorid，marchid，mimpid の読み出しには，ゲストに提示する定数を宛先レジスタへ格納する命令列を生成する．mideleg と medeleg は，ゲスト S-mode へトラップを委譲しない構成に合わせて 0 を返す．mhartid は，VCPU ごとの専用データ領域に保持した識別子を読み出す．

4.4.3　値変換

実 CSR とゲストに提示する CSR の間で，ビット配置や Exception Code フィールドの値が異なる場合には，読み出し・書き込みに値変換を組み込む．ビットフィールドの変換には図 2〜4，mcause の変換には表 1 の対応を用いる．従来 Violet がトラップ処理内で行っていた変換を，VM 内で実行する生成コードによって実現する．

値変換では，CSR の値によって実行命令数と制御フローが変化しないように命令列を生成する．変換対象のビット位置と CSR 間の値の対応関係は生成時に確定するため，それらを命令中の定数として埋め込む．一方，変換する CSR 値は実行時に取得する．ビットフィールドの再配置にはシフトとマスク演算を，mcause の変換には一致判定とマスク演算を用いる．

ビットフィールドの抽出と再配置

入力値を x，変換元の開始ビット位置を p，変換先の開始ビット位置を q，フィールド幅を w とする．変換先のビット列 t は，式 (1) によって求める．

$$t = \big((x \gg p)\ \&\ (2^w - 1)\big) \ll q \tag{1}$$

ここで，$\gg$ は論理右シフト，$\ll$ は左シフト，& はビットごとの論理積を表す．出力を 0 で初期化し，各フィールドについて求めた t を論理和で合成する．生成する命令列は，SRLI による抽出位置の移動，ANDI によるマスク，SLLI による再配置，OR による合成からなる．

例えば，mie への書き込みでは，(p, q, w) が $(3, 1, 1)$，$(7, 5, 1)$，$(11, 9, 1)$ となる 3 組について，式 (1) の命令列を生成する．入力の各ビットが 0 であるか 1 であるかにかかわらず，3 組すべてに同じ演算を適用する．読み出し時は p と q を入れ替え，vsie の値からゲストに返す mie の値を組み立てる．mip と vsip の間にも同じ処理を適用する．

一致判定とマスクによる mcause の値変換

mcause の変換では，表 1 の各組について，入力が変換元の値と一致するかを判定する．入力を x，変換元の値を a_i，変換先の値を b_i とする．一致時に 1，不一致時に 0 となる値 e_i と，64 ビットのマスク m_i を，式 (2) によって求める．

$$e_i = \begin{cases} 1 & (x = a_i), \\ 0 & (x \neq a_i), \end{cases} \qquad m_i = 0 - e_i \pmod{2^{64}} \tag{2}$$

一致時には m_i の全ビットが 1 となり，不一致時には全ビットが 0 となる．出力 y を x で初期化し，各組について式

(3) に従って更新する．

$$y \leftarrow (y \,\&\, \neg m_i) \mid (b_i \,\&\, m_i) \qquad (3)$$

¬ はビット反転，| はビットごとの論理和を表す．一致した組では y を b_i に置き換え，不一致の組では y を保持する．いずれの組にも一致しない場合は，入力値をそのまま返す．

一致判定も条件分岐を使わずに行う．2 値の排他的論理和は，一致するときだけ 0 になる．この結果を符号なし整数として 1 と比較すれば，一致時に 1，不一致時に 0 が得られる．これを 0 から減算して全ビットが 1 または 0 のマスクを作り，式 (3) へ適用する．生成コードでは，最上位の Interrupt ビットと下位 63 ビットの Exception Code フィールドをそれぞれ比較し，両方の一致を表すマスクの論理積を取る．

例えば，読み出した値が $I+5$ であればタイマ割込みの組だけが一致し，$I+7$ を返す．一方，Exception Code フィールドの値が同じ 5 でも，Interrupt ビットが 0 の例外はこの組に一致せず，値 5 を保持する．このように，Exception Code フィールドと Interrupt ビットを併せて判定することで，割込みと例外を正しく変換する．書き込み時には a_i と b_i を入れ替えて逆方向の変換を行う．

4 組の比較と選択はコード生成時にすべて展開し，一致後も残りの組に対する演算を実行する．各組の比較には変換前の入力 x を用いるため，置換済みの出力が別の変換対象になることもない．実行時の表探索，ループ，条件分岐を使わず，入力値によらず一定の命令列を実行することで，入力値や分岐予測に起因するジッタを抑える．

4.5 トラップ処理と MRET エミュレーションの実装

ゲストの `mepc` と `mstatus` の割込み許可状態は，シャドウ CSR で管理している．トラップ発生時にハードウェアが更新するのは実 CSR であるため，そのままゲストのハンドラを実行すると，ゲストが参照するシャドウ CSR にトラップ発生時の状態が反映されない．そこで，ハンドラの実行前にシャドウ CSR を更新するため，`stvec` が指すトラップエントリをシャドウ CSR 同期用コードに差し替える．これに伴い，ゲストが指定するトラップエントリはシャドウ `mtvec` に保持する．

シャドウ CSR 同期用コードでは，`sepc` の値をシャドウ `mepc` へ保存する．また，シャドウ `mstatus` の MIE を MPIE へ保存し，MIE を 0 にする．その後，シャドウ `mtvec` に従って，割込みを禁止した状態でゲストのハンドラへ制御を移す．

MRET エミュレーションでは，割込み禁止とレジスタ退避の後，シャドウ `mstatus` の MPIE を MIE へ戻し，MPIE を 1 にする．後処理には，復帰先としてシャドウ `mepc`，割込み許可状態として更新後の MIE を渡す．擬似コード 2 に，トラップ発生時と復帰時の状態更新を示す．前処理とレジスタの退避は省略し，後処理は擬似コード 1 の ResumeGuest で表す．

擬似コード 2 トラップ発生時と `MRET` の状態更新

```
 1: procedure TrapEntry(pc, cause)
 2:     S.mepc ← pc
 3:     S.mstatus.MPIE ← S.mstatus.MIE
 4:     S.mstatus.MIE ← 0
 5:     target ← HandlerAddress(S.mtvec, cause)
 6:     ResumeGuest(target, 0)
 7: end procedure

 8: procedure Mret
 9:     S.mstatus.MIE ← S.mstatus.MPIE
10:     S.mstatus.MPIE ← 1
11:     ResumeGuest(S.mepc, S.mstatus.MIE)
12: end procedure
```

表 3 トラップ時の制御移行先

モード	例外	割込み
Direct	`base`	`base`
Vectored	`base`	`base` + 4 × `cause`

HandlerAddress は，`mtvec` に設定されたベースアドレスと Direct ／ Vectored モードに従って，ゲストのハンドラのアドレスを求める．表 3 に，モードごとの制御移行先を示す．`base` はベースアドレス，`cause` は Exception Code フィールドの値を表す [6]．

各モードに対応する処理のコードをあらかじめ生成し，ゲストが `mtvec` を更新した時点で，対応するコードの先頭アドレスを `stvec` へ設定する．この選択にはマスク演算を用いる．トラップ発生時には選択済みのコードを実行するため，モード判定を省略できる．

4.6 コード配置と命令の置換

命令の置換によって後続命令のアドレスが変わらないよう，32 ビット長の CSR アクセス命令や `MRET` を，同じ長さの命令に置換する．直接置換では CSR 番号のみを変更し，エミュレーションコードへ制御を移す場合は，戻りアドレスをレジスタに保存しないジャンプ命令 `jal x0, offset` へ書き換える．

生成したエミュレーションコードは，VM 内に確保した 16 KiB の保存領域（コードキャッシュ）に格納する．`JAL` 命令は 1 命令で即値を用いたジャンプが可能であるが，ジャンプ元からの相対距離で行き先を指定するため，到達範囲は前後約 1 MiB に限られる．このため，コードキャッシュを変換対象の命令の近傍に配置する．CSR エミュレーションと MRET エミュレーションからの復帰には，`sepc` に設定したアドレスへ制御を移す `SRET` を使うため，同じ

ⓒ2026 Information Processing Society of Japan

距離の制約は生じない．

コードを書き換えた後は，プロセッサが更新後の命令を読み出せるようにする必要がある．メモリへの書き込みだけでは，命令キャッシュなどに残る更新前の命令が実行される可能性がある．`FENCE.I` 命令は，先行するメモリ書き込みを，同じハードウェアスレッド（hart）で後続の命令フェッチに反映させる同期命令である [10]．本実装では，コード生成と元命令の置換後にこの同期を行い，元の PC からゲストを再開する．

4.7 キャッシュカラーリングの適用

生成コード，シャドウ CSR，専用スタックには，ゲスト OS とは異なるキャッシュカラーを割り当てる．さらに，これらを指すページテーブルも同じカラーに配置する．TLB ミス時のページテーブルウォークもメモリアクセスを伴うため [6]，命令フェッチやデータアクセスと同様にキャッシュ競合の抑制対象とする．

評価機に搭載されたプロセッサの L2 キャッシュは 256 KiB，8-way であり [11]，ページサイズを 4 KiB とすると，カラー数は

$$N_{\mathrm{color}} = \frac{256\,\mathrm{KiB}}{8 \times 4\,\mathrm{KiB}} = 8$$

となる．物理アドレスの連続したビットを L2 キャッシュのセットインデックスに用いる構成を仮定すると，カラーはホスト物理アドレスのビット 12–14 によって決まる．同じカラーの物理ページは 32 KiB 間隔で配置される．生成コードと専用データ領域には 1 カラーを割り当て，ゲスト OS のコード・データには異なるカラーを割り当てる．物理ページは非連続になるが，ゲスト物理アドレス空間では連続した領域としてマッピングされる．

さらに，生成コードや専用データ領域を指すページテーブルにも，同じカラーの物理ページを割り当てる．そのために，ゲスト物理アドレスの範囲ごとにページテーブル用の物理ページを予約し，ページテーブル構築時に対象範囲に対応するページを使用する．

生成コードと専用データ領域を指すページテーブルの枚数を抑えるため，これらを同一の 2 MiB 境界内に配置する．本実装で用いる 4 段のゲスト物理アドレス変換（Sv48x4）では，この配置によってルート以外のページテーブルを 3 枚でまとめられる．ページテーブル 12 KiB，専用データ領域 4 KiB，コードキャッシュ 16 KiB の合計は 32 KiB となり，1 カラー分のキャッシュ容量に相当する．対象は合計 8 枚の 4 KiB ページであり，各キャッシュセットに対応するライン数が 8-way の範囲に収まるため，これらを同時に L2 キャッシュへ保持できる．ゲスト OS には異なるカラーを割り当てる．この配置により，ゲスト OS のメモリアクセスによる生成コードと専用データ領域の L2 キャッシュからの追い出しを防ぐ．

表 4　比較する実行構成

表記	構成
Baremetal	ハイパーバイザなしの M-mode 実行
Violet-T	トラップ&エミュレート方式
Violet-DBT	動的バイナリ変換
Violet-DBT-C	動的バイナリ変換＋カラーリング

```
# Read                  # Write
rdcycle t0              rdcycle t0
csrr    t2, csr         csrw    csr, t2
rdcycle t1              rdcycle t1
sub     t3, t1, t0      sub     t3, t1, t0
```

図 9　CSR アクセス時間の測定

5. 評価

提案方式を，定常実行時の CSR アクセス時間，実行時間変動，RTOS のスループット，初回実行時間と生成コードサイズの観点から評価する．

5.1 評価環境と測定方法

評価には SiFive HiFive Premier P550 を用いる．搭載 SoC は ESWIN EIC7700X，プロセッサは SiFive P550，動作周波数は 1.4 GHz である．RTOS の評価には FreeRTOS を用いる．表 4 に，比較対象を示す．ベアメタル実行，従来のトラップ&エミュレート方式，動的バイナリ変換のみを適用した構成，動的バイナリ変換とキャッシュカラーリングを適用した構成の 4 種類を比較する．

CSR アクセス時間は，対象命令の直前と直後に `cycle` を読み出し，その差分 $\Delta C = C_{\mathrm{after}} - C_{\mathrm{before}}$ として求める．対象命令のオペランドには，カウンタの読み出しに用いるレジスタとは別のレジスタを使用する．読み出しと書き込みは個別に測定する．測定値には，カウンタ読み出しによるオーバヘッドを含む．

CSR ベンチマークには 4 構成で同一のゲストプログラムを用いる．15 種類の CSR の読み出しと，読み書き可能な 11 種類の CSR の書き込みをそれぞれ 1000 万回測定する．図 9 に，1 回分の測定を示す．図中の `csr` は測定対象の CSR を表す．

初回のコード生成と命令の置換を含む先頭 10 回をウォームアップとして除外し，残る 9,999,990 回の最小値，平均値，最大値を求める．この除外条件は全構成・全操作で共通とする．以下では，この測定範囲における最大値を最大実行時間とし，最大値と最小値の差をジッタとする．

5.2 定常実行時の CSR アクセス時間

表 5 と表 6 に，各 CSR の読み出し時間と書き込み時間を示す．

ⓒ2026 Information Processing Society of Japan

直接置換の例として，mtval では，Violet-DBT と Violet-DBT-C の読み書き時間はいずれも 14 サイクルで一定となり，ベアメタルと同じ実行時間を得た．Violet-T の最大実行時間は読み出しで 1814 サイクル，書き込みで 1818 サイクルであり，直接置換によって約 130 分の 1 に短縮した．

個別に生成したコードでエミュレートする CSR では，Violet-DBT の平均実行時間は 122.0–167.0 サイクルとなり，Violet-T の 1094.7–1137.5 サイクルから短縮した．最大実行時間は 715–1055 サイクルであり，各 CSR について Violet-T の約 1.7–2.5 分の 1 となった．

5.3 キャッシュカラーリングによる変動抑制

Violet-DBT と Violet-DBT-C を比較し，生成コード，専用データ領域，それらを指すページテーブルへのカラーリングの効果を評価する．例えば，mstatus の書き込み時間は，Violet-DBT では 151–1052 サイクル，Violet-DBT-C では 151–154 サイクルとなり，ジッタは 901 サイクルから 3 サイクルに縮小した．両構成の平均実行時間は約 151 サイクルであり，カラーリングによって最大実行時間を低減した．Violet-DBT-C では，評価したすべての CSR アクセスのジッタが 4 サイクル以下となった．個別に生成したコードでエミュレートする CSR の最大実行時間は 122–167 サイクルであり，動的バイナリ変換と合わせて，Violet-T の約 11–15 分の 1 に短縮した．例えば，mtvec の書き込みの最大実行時間は，Violet-T の 1824 サイクル，Violet-DBT の 1055 サイクルに対し，Violet-DBT-C では 167 サイクルであった．これらの結果から，キャッシュカラーリングの適用により，生成コードと専用データ領域の L2 キャッシュからの追い出しを抑制し，CSR アクセス時間の変動を低減できることが分かる．

5.4 RTOS のスループット

RTOS 向けベンチマークスイート Thread-Metric の Cooperative Scheduling Test を FreeRTOS 上で実行した [12]．このテストでは，同一優先度の 5 タスクが自発的なタスク切り替えを繰り返す．各タスクは実行再開後にカウンタを加算し，30 秒間における全タスクのカウンタ増分の合計をスコアとする．表 7 に結果を示す．

Violet-DBT は Violet-T の約 12.1 倍，Violet-DBT-C は約 11.6 倍のスコアを得た．複数の CSR 操作を伴うタスク切り替え処理に対しても，動的バイナリ変換による性能向上を確認した．一方，Violet-DBT-C のスコアは Baremetal の約 6.0%であり，ベアメタルとの性能差は残る．

カラーリングの適用により，スコアは 11,653,041 から 11,222,351 へ約 3.7%低下した．表中の換算時間は，タスクの実行と切り替えを含む 1 回あたりの平均時間を表す．

5.5 初回実行時間と生成コードサイズ

mcause 読み出しの初回実行時間は，Violet-DBT で 127,353 サイクル，Violet-DBT-C で 119,070 サイクルであった．動作周波数 1.4 GHz における実行時間は，それぞれ約 91.0 μs，約 85.1 μs である．一方，定常実行時の平均読み出し時間は両構成とも 159.0 サイクル（約 0.114 μs）であった（表 5）．初回の測定区間には，トラップ処理，命令解析，コード生成，命令の置換，生成コードの実行を含む．定常実行時とは異なり，初回実行時にはコード生成などによる大きな遅延が生じる．

CSR エミュレーション用のコードのサイズは，CSR の処理方式，オペランド，復帰先アドレスによって異なる．csrrw x0，mepc，a0 について，復帰先を 0x80001000 とした場合の生成コードは，前処理と後処理を含めて 30 命令，120 バイトである．トラップ発生時に実行するコードは，Direct モード用が 124 バイト，Vectored モード用が 176 バイト，予約モード用が 4 バイトである．MRET エミュレーション用のコードは 136 バイトであり，これらの合計は 440 バイトとなる．CSR エミュレーション用のコードは変換した命令位置の数に応じて増加する一方，VM 初期化時に生成するこれらのコードのサイズは，命令位置の数に依存しない．

6. 議論と制約

6.1 タスクの起動周期と処理時間

Cooperative Scheduling Test における，タスクの実行と切り替えを含む 1 反復あたりの平均時間は，Violet-T で 31.039 μs，Violet-DBT-C で 2.673 μs であった（表 7）．例えば，100 μs の起動周期に対して，これらの平均時間はそれぞれ約 31.0%，約 2.7%に相当する．同テストの 1 反復相当の処理を各周期で実行する場合，提案方式によって，周期内でこの処理が占める時間を小さくできる．

ただし，周期処理の期限達成を判断するには，平均時間だけでなく，初回実行時の遅延や他の処理・干渉を含む最大応答時間を評価する必要がある．

6.2 初回実行時のオーバヘッド

提案方式では，各命令の初回実行時に命令解析，コード生成，命令の置換に伴うオーバヘッドが生じる．割込みハンドラ内の未変換命令を初めて実行する場合にも，この遅延が割込み処理時間に加わる．本評価では初回実行時間を測定したが，RTOS の動作や時間制約への影響は未評価であり，その評価が今後の課題である．

ゲストイメージと実行時のコード配置を事前に確定できる組込みシステムでは，対象命令をあらかじめ変換する Ahead-of-Time（AOT）変換も，初回遅延を避ける選択肢となる．AOT 変換では，ゲストバイナリの命令列を解析し，変換対象の命令位置を特定する必要がある．提案方式

ⓒ2026 Information Processing Society of Japan

表 5　CSR 読み出し時間（サイクル）

	Baremetal				Violet-T				Violet-DBT				Violet-DBT-C			
CSR	平均	最小	最大	ジッタ	平均	最小	最大	ジッタ	平均	最小	最大	ジッタ	平均	最小	最大	ジッタ
mstatus	14.0	14	14	0	1131.1	1112	1830	718	141.0	141	946	805	141.0	141	141	0
mie	14.0	14	14	0	1131.8	1114	1804	690	137.0	137	894	757	137.0	137	137	0
mip	14.0	14	14	0	1131.5	1114	1780	666	137.0	137	1031	894	137.0	137	137	0
mtvec	14.0	14	14	0	1137.5	1128	1847	719	163.0	162	1049	887	163.0	162	166	4
mepc	14.0	14	14	0	1110.7	1094	1822	728	125.0	125	896	771	125.0	125	129	4
mcause	14.0	14	14	0	1111.8	1095	1788	693	159.0	159	1030	871	159.0	159	159	0
mtval	14.0	14	14	0	1111.8	1108	1814	706	14.0	14	14	0	14.0	14	14	0
mscratch	14.0	14	14	0	1115.8	1107	1802	695	125.0	125	818	693	125.0	125	126	1
medeleg	14.0	14	14	0	1114.7	1087	1764	677	122.0	122	883	761	122.0	122	125	3
mideleg	14.0	14	14	0	1115.8	1107	1860	753	122.0	122	931	809	122.0	122	125	3
misa	14.0	14	14	0	1118.8	1089	1804	715	122.0	122	853	731	122.0	122	125	3
mhartid	14.0	14	14	0	1097.4	1071	1760	689	125.0	125	881	756	125.0	125	126	1
mvendorid	14.0	14	14	0	1095.6	1077	1788	711	122.0	122	775	653	122.0	122	122	0
marchid	14.0	14	14	0	1094.7	1079	1757	678	122.0	122	873	751	122.0	122	125	3
mimpid	14.0	14	14	0	1095.6	1092	1794	702	122.0	122	929	807	122.0	122	122	0

表 6　CSR 書き込み時間（サイクル）

	Baremetal				Violet-T				Violet-DBT				Violet-DBT-C			
CSR	平均	最小	最大	ジッタ	平均	最小	最大	ジッタ	平均	最小	最大	ジッタ	平均	最小	最大	ジッタ
mstatus	22.0	22	22	0	1130.7	1102	1820	718	151.0	151	1052	901	151.0	151	154	3
mie	22.0	22	22	0	1133.8	1099	1836	737	149.0	149	918	769	149.0	149	149	0
mip	22.0	22	22	0	1131.6	1105	1814	709	149.0	149	964	815	149.0	149	150	1
mtvec	22.0	22	22	0	1134.7	1117	1824	707	167.0	164	1055	891	167.0	167	167	0
mepc	14.0	14	14	0	1110.8	1094	1764	670	124.0	124	930	806	124.0	124	128	4
mcause	14.0	14	14	0	1110.0	1099	1770	671	156.0	156	1023	867	156.0	156	156	0
mtval	14.0	14	14	0	1112.6	1094	1818	724	14.0	14	14	0	14.0	14	14	0
mscratch	14.0	14	14	0	1118.8	1101	1810	709	124.0	124	898	774	124.0	124	124	0
medeleg	22.0	22	22	0	1113.5	1097	1775	678	122.0	122	715	593	122.0	122	125	3
mideleg	22.0	22	22	0	1111.8	1103	1798	695	122.0	122	771	649	122.0	122	125	3
misa	22.0	22	22	0	1116.9	1101	1812	711	122.0	122	747	625	122.0	122	122	0

表 7　Cooperative Scheduling Test の結果

構成	スコア	換算時間（μs）
Baremetal	186,968,445	0.160
Violet-T	966,529	31.039
Violet-DBT	11,653,041	2.574
Violet-DBT-C	11,222,351	2.673

では，対象命令の実行時に必ず例外が発生することを利用し，例外発生位置から変換すべき命令を確実に特定できる．自己書き換えコードへの対応を拡張した際には，AOT 変換後に書き換えられた命令を実行時に捕捉・変換するフォールバックとしても利用できる．初回遅延への対策として，初期化時の mtvec 設定を契機に，指定されたトラップハンドラを起点として命令列を事前スキャンし，対象命令の変換と置換を済ませる方法が考えられる．これにより，割込み処理中に生じる初回変換のオーバヘッドを初期化時へ移せる．適用にあたっては，ハンドラから呼び出される処理も含めたスキャン範囲と，事前変換に要する初期化時間を評価する必要がある．

6.3　適用範囲と生成コードの制約

提案方式は，M-mode のみを使用するゲスト OS を対象とし，VM 内の S-mode CSR を用いて M-mode CSR へのアクセスをエミュレートする．そのため，ゲスト内で M-mode と S-mode を併用する構成には，そのまま適用できない．また，ゲストの命令列を書き換えられることと，

コードキャッシュが JAL の到達範囲内にあることを前提とする．広いアドレス範囲にコードを配置するゲストへ適用するには，コード領域ごとに近傍のコードキャッシュを設けるなどの拡張が必要となる．また，CSR エミュレーション用のコードは命令位置ごとに生成するため，コードキャッシュの必要容量は変換対象の命令位置の数と処理内容に依存する．容量不足時の処理と，その際の実行時間変動の評価が課題となる．自己書き換えコードに対応するには，書き換え箇所に対応する生成コードを無効化し，更新後の対象命令を再び置換する仕組みが必要となる．複数 hart でコードを共有する場合には，命令置換時の排他制御と，各 hart の命令キャッシュの同期が必要である．

6.4 ゲストの権限と隔離

シャドウ CSR はゲスト自身の M-mode の状態を保持し，エミュレーションコードはゲストが本来実行できる M-mode 操作を代行する．M-mode のゲスト OS は自身の CSR と実行コードを管理する権限を持つため，これらを同じゲストアドレス空間に配置することは，本来の権限関係に違反しない．エミュレーションコードもゲストと同じ VS-mode で実行され，他の VM およびハイパーバイザとの隔離には，引き続き Hypervisor 拡張の特権分離とゲスト物理アドレス変換を用いる [6]．この配置でも，VM とハイパーバイザの間および VM 間の保護境界を維持する．

6.5 スループットと実行時間変動のトレードオフ

キャッシュカラーリングは競合を抑制する一方，ゲストが利用できるキャッシュ容量と物理ページを制限する．Cooperative Scheduling Test では，カラーリング適用後のスコアが約 3.7%低下した．実行時間変動の抑制とスループットの両立には，ゲストのワーキングセットと生成コードサイズを考慮してカラーを割り当てる必要がある．

条件分岐をマスク演算へ置き換えると，比較結果にかかわらず選択処理の命令列を実行する．また，繰り返し処理の展開は，実行時のループ制御を削減する一方で，生成コードサイズを増加させる．この設計では，最短の実行経路やコードサイズだけでなく，入力値によって実行経路が変化しないことを重視している．

6.6 割込み配送と残存するオーバヘッド

従来方式におけるタイマ割込みの遅延時間は，平均がベアメタルで 0.16 μs，Violet で 1.05 μs，最大がそれぞれ 0.52，1.31 μs であった [5]．提案方式による割込み遅延時間の短縮効果は，別途評価する必要がある．

RTOS 全体の応答時間には，CSR エミュレーションに加え，CLINT や Platform-Level Interrupt Controller (PLIC) へのアクセス，割込み配送，ホスト側の処理が影響する．

7. 関連研究

7.1 RISC-V における RTOS の仮想化

Bao の RISC-V 対応では，Hypervisor 拡張を用いてゲスト OS を VS-mode 上で実行する [13]．Bao 向けの FreeRTOS は，S-mode の実行環境に合わせてアーキテクチャ依存部を変更している [14]．具体的には，割込み制御に sstatus を用い，例外復帰を SRET で行い，タイマを SBI 経由で設定する．この方式では，M-mode 向け FreeRTOS をゲストとして実行するために，OS 側の CSR 操作や特権命令，タイマ制御を変更する必要がある．

HSP-V は，Hypervisor 拡張を持たない RISC-V プラットフォームにおいて，OpenSBI の Domain 機構と PMP を用いて実行環境を分離する [15]．各パーティションは S-mode または U-mode で実行されるため，M-mode 向け RTOS を実行するには，その特権操作を実行環境に適合させる必要がある．

7.2 動的バイナリ変換

Agesen らは，VM exit（ゲストからハイパーバイザへの遷移）を起こした命令に続く命令列を解析し，複数の命令をまとめて処理する変換コードを生成する [16]．変換範囲には，VM exit を起こす命令の間にある通常の命令も含める．各実行時には，先頭の対象命令で VM exit し，ハイパーバイザ側で変換コードを実行することで，後続の命令による VM exit を削減する．提案方式では，初回実行時の例外で対象命令を捕捉し，その命令を直接書き換えるか，生成したエミュレーションコードへのジャンプに置換する．以後はゲスト側で置換後の処理を実行するため，定常時の対象命令のエミュレーションには VM exit が発生しない．Agesen らが 1 回の VM exit で処理する命令の範囲を広げるのに対し，提案方式は命令位置ごとの置換によって，定常時の VM exit そのものを除去する．

Mittal らは，Power Architecture（PowerPC 系）の組込みプロセッサを対象に，特権命令による VM exit を削減する in-place binary translation を提案した [17]．VM exit を頻繁に起こす命令を，対応する 1 命令，またはゲストアドレス空間に配置した変換コードへの分岐命令に置換する．複数命令の変換コードは命令位置ごとに生成し，実行後は元命令の次へ戻る．命令単位で元のコードを書き換え，変換後の処理をゲスト内で実行する点は提案方式と共通する．Mittal らは，コード領域の確保と自己書き換えへの対応に読み書きの監視を用い，監視に伴うページフォルトと TLB ミスを削減する．提案方式は，RISC-V の M-mode 仮想化における CSR エミュレーションを対象とし，条件分岐と実行時のループを省いた命令列と，生成コード・専用データ領域・ページテーブルのキャッシュカラーリングによっ

ⓒ2026 Information Processing Society of Japan

てジッタを抑える．

7.3 キャッシュ分離

共有キャッシュの競合抑制には，物理ページの割り当てによって使用するキャッシュセットを分けるキャッシュカラーリングが用いられる．仮想化環境への適用例として，Xen 上のページカラーリングに基づくキャッシュ分割 [18] や，Bao の VM ごとのカラー割り当て [4] がある．提案方式では，キャッシュカラーリングを同一 VM 内のゲスト OS とエミュレーション処理の分離に適用する．生成コード，専用データ領域，それらを指すページテーブルを対象とし，ゲスト OS のアクセスによる L2 キャッシュからの追い出しを防ぐことで，エミュレーションの実行時間変動を抑制する．

8. おわりに

本論文では，RISC-V 向けハイパーバイザ Violet の M-mode エミュレーションを高速化し，ジッタを低減する方式を提案した．命令単位の動的バイナリ変換により，命令置換後の処理を VM 内で完結させ，定常時のハイパーバイザへのトラップを不要にした．また，生成したエミュレーションコードから条件分岐と実行時のループを排除し，キャッシュカラーリングによってゲスト OS とのキャッシュ干渉を抑えることで，実行時間の変動を低減した．定常時の CSR アクセスでは，最大実行時間が直接置換で従来方式の約 130 分の 1，生成コードの実行で約 11–15 分の 1 となった．対象となる全 CSR アクセスのジッタも 4 サイクル以下に抑え，高速化と低ジッタ化を実現した．さらに，FreeRTOS 上で実行した Thread-Metric の Cooperative Scheduling Test では，従来方式の約 11.6 倍のスループットを得た．

今後の課題は，初回実行時のオーバヘッドが RTOS の動作に与える影響の評価と，コードキャッシュの容量と配置に関する制約の緩和である．

参考文献

[1] Patel, A., Daftedar, M., Shalan, M. and El-Kharashi, M. W.: Embedded Hypervisor Xvisor: A Comparative Analysis, *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 682–691 (online), DOI: 10.1109/PDP.2015.108 (2015).

[2] Li, H., Xu, X., Ren, J. and Dong, Y.: ACRN: A Big Little Hypervisor for IoT Development, *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pp. 31–44 (online), DOI: 10.1145/3313808.3313816 (2019).

[3] Reinhardt, D. and Morgan, G.: An Embedded Hypervisor for Safety-Relevant Automotive E/E-Systems, *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pp. 189–198 (online), DOI: 10.1109/SIES.2014.6871203 (2014).

[4] Martins, J., Tavares, A., Solieri, M., Bertogna, M. and Pinto, S.: Bao: A Lightweight Static Partitioning Hypervisor for Modern Multi-Core Embedded Systems, *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, Open Access Series in Informatics (OASIcs), Vol. 77, pp. 3:1–3:14 (online), DOI: 10.4230/OASIcs.NG-RES.2020.3 (2020).

[5] Kito, T., Yamamoto, R., Horii, K., Masuda, H. and Mouri, K.: Violet: Enabling Full Virtualization for M-mode RTOS on RISC-V, arXiv:2609.09833 (2026).

[6] RISC-V International: *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 20240411* (2024).

[7] 山本遼介，追立真吾：RISC-V アーキテクチャにおける仮想マシン上での RTOS 動作実現方式検討と試作，情報処理学会研究報告， Vol. 2024-ARC-256, No. 5, pp. 1–7 (2024).

[8] 鬼頭太郎, 山本遼介, 堀井圭祐, 追立真吾, 毛利公一：RISC-V における RTOS を実行可能なハイパーバイザ Violet の実機移植，研究報告システムソフトウェアとオペレーティング・システム（OS）， Vol. 2025-OS-167, No. 8, pp. 1–9 (2025).

[9] RISC-V International: *The RISC-V Instruction Set Manual, Volume I: Unprivileged Architecture, Version 20240411, Zicsr Extension* (2024).

[10] RISC-V International: *The RISC-V Instruction Set Manual, Volume I: Unprivileged Architecture, Version 20240411, Zifencei Extension* (2024).

[11] Beijing ESWIN Computing Technology Co., Ltd.: *EIC7700X SOC Manual, Version 1.0* (2024).

[12] Eclipse ThreadX: Thread-Metric Test Suite, `https://github.com/eclipse-threadx/threadx/tree/master/utility/benchmarks/thread_metric`.

[13] Sá, B., Martins, J. and Pinto, S.: A First Look at RISC-V Virtualization From an Embedded Systems Perspective, *IEEE Transactions on Computers*, Vol. 71, No. 9, pp. 2177–2190 (online), DOI: 10.1109/TC.2021.3124320 (2022).

[14] Bao Project: freertos-over-bao, `https://github.com/bao-project/freertos-over-bao`.

[15] Sousa, J. a., Martins, J., Gomes, T. and Pinto, S.: HSP-V: Hypervisor-Less Static Partitioning for RISC-V COTS Platforms, *IEEE Access*, Vol. 12, pp. 71131–71144 (online), DOI: 10.1109/ACCESS.2024.3399601 (2024).

[16] Agesen, O., Mattson, J., Rugina, R. and Sheldon, J.: Software Techniques for Avoiding Hardware Virtualization Exits, *USENIX Annual Technical Conference*, pp. 373–385 (2012).

[17] Mittal, A., Bansal, D., Bansal, S. and Sethi, V.: Efficient Virtualization on Embedded Power Architecture Platforms, *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 445–458 (online), DOI: 10.1145/2451116.2451163 (2013).

[18] Wang, X., Wen, X., Li, Y., Luo, Y., Li, X. and Wang, Z.: A Dynamic Cache Partitioning Mechanism under Virtualization Environment, *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 1907–1911 (online), DOI: 10.1109/TrustCom.2012.25 (2012).

ⓒ2026 Information Processing Society of Japan