論　文

産業機器のエッジ制御のための“LAXER-SoC”用CPU“松竹V”

松永　大輝[†]　　竹岡　尚三[†]

”Shochiku-V”: CPU of ”LAXER-SoC” for industrial edge devices

Daiki MATSUNAGA[†] and Shozo TAKEOKA[†]

あらまし　本研究では，エッジ制御用 SoC に搭載するために強化した RISC-V CPU を開発した．本 CPU は，通常はソフトウェアで実装されるタスク・スケジューラやセマフォなどの実時間 OS 相当の機能をハードウェアのみで実現した．割込み機構の代替として，ハードウェア・セマフォを用いた外部イベント通知機構を持つ．産業機器やロボットなどのエッジ制御に使用するチップは，低消費電力とコストの要請から，搭載するメモリ量が小さくなる傾向がある．実時間 OS 相当の機能をハードウェアで実現することで，メモリ・フットプリントを削減でき，タスク切り替え等のオーバヘッドも小さくできた．ハードウェアによる実現であるため，OS 相当機能は堅牢でもある．また，エッジ AI アプリケーションの高速化のため，8-bit 浮動小数点数のベクトル演算命令や，多方向ジャンプ命令を追加した．本 CPU を搭載した論理を，TSMC 90nm プロセスで製造し 920 万トランジスタの SoC として完成した．

キーワード　CPU, RTOS, RISC-V, ASIC, Prolog

1. はじめに

我々は，RISC-V CPU の拡張として，タスク・スケジューラやディスパッチャ，セマフォなどの実時間 OS 相当の機能をハードウェアにて実現し，加えて，エッジ AI アプリケーションの高速化のため，多方向ジャンプ命令や 8-bit 浮動小数点数ベクトル演算命令を追加した CPU”松竹 V”を開発した．

松竹 V は，産業機器やロボットなどのエッジにおける制御のための SoC である”LAXER-SoC”へ搭載するために開発した．エッジ制御においては，ロボット向けミドルウェアの通信，判断のための AI 推論の高速な実行，モータなどの制御信号を生成などする機能が求められる．本 SoC は，ロボット分野で広く用いられているミドルウェアである ROS2 の通信を高速かつ低消費電力で実現するため，その通信プロトコルをハードウェアで実現した．これにより，CPU の介在なしで本 SoC は ROS2 ノードになることができる．また，信号生成のために，任意波形生成器と PWM を持ち，サーボモータなどの制御が可能である．低消費電力を実現するため，動的にクロック周波数の変更を行うことができる．本 CPU は，LAXER-SoC の周辺装置の制御，AI 推論などの実行を行う．

[†] 株式会社アックス
AXE, Inc.
DOI:10.14923/transj.???????????

2. 背　　景

2.1 実時間 OS のオーバヘッド

エッジ制御の SoC は，低消費電力であることや，低コストでの実現が求められる．したがって，大容量のメモリや消費電力の大きな高速 CPU を搭載することが難しい．

組み込みシステムにおいては，マルチタスクを使用したアプリケーション開発を容易にするため，実時間 OS が使用されることが多い．しかし，実時間 OS は ROM, RAM と CPU 時間を消費する．タスク切り替えやセマフォ操作などは，レジスタの退避・復元やスケジューリング処理をソフトウェアで行い, 時間的オーバヘッドを生じさせる．

本 CPU では，タスク・スケジューラやディスパッチャをハードウェアで実現することで，実時間 OS が必要とするメモリ・フットプリントを削減した．実時間 OS 相当の機能の実現に，ほとんど ROM を必要とせず，RAM も各タスクごとの仕事が本来必要な領域だけがあればよい．

実時間 OS に相当する機能をハードウェアで実現することで，時間的オーバヘッドを削減できた．レジスタセットをタスクの数だけ持つため，レジスタ退避と復元が不要であり，最短で 1 サイクルごとにタスク切り替えを行うことが可能である．また，実時間 OS 相当機能をすべてハードウェアで実現したことから，堅牢かつ高セキュアでもある．

ハードウェア・セマフォは，タスクの排他制御だけでなく，割り込みの代替となる外部イベント機構において同期に使用される．

2.2 論理推論 AI

論理推論 AI は，論理式で記述された知識に基づいて推論を行う．我々は長年，帰納論理推論の実行系である Prolog 言語を用いて，推論システムを構築している．

動的な高級言語実装の定石として，データ・オブジェクトなどの動的な型判定が行われる．その際にオブジェクトの型によって多方向のジャンプを行う．オブジェクトの型によってジャンプ先を決定する GNU Prolog のランタイム関数 `PL_Switch_On_Term()` を，図 1 に示す．多方向ジャンプを行う箇所では，`PL_Switch_On_Term()` 関数を呼び出したのち，返り値のアドレスにジャンプする．この手続きは，極めて頻繁に実行されるため，高速化が望まれる．

図 1 GNU Prolog のランタイム関数における多方向ジャンプ

```
CodePtr FC Pl_Switch_On_Term(CodePtr c_var,
  CodePtr c_atm, CodePtr c_int,
  CodePtr c_lst, CodePtr c_stc)
{
  ...
  if (tag_mask == TAG_INT_MASK)
    codep = c_int;
  else if (tag_mask == TAG_ATM_MASK)
    codep = c_atm;
  else if (tag_mask == TAG_LST_MASK)
    codep = c_lst;
  else if (tag_mask == TAG_STC_MASK)
    codep = c_stc;
  else
    codep = c_var;

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

2.3 機械学習 AI

最近では，エッジ制御では，ディープラーニングなど機械学習（ML）AI の推論を高速に行うことが望まれる．ML AI の推論では，量子化などで重み等の精度を減少させても，推論精度の低下は小さいと報告されている [1], [2]．そこで，本 CPU では ML AI の推論の高速化のため，エッジ機器にふさわしい精度である 8-bit 浮動小数点数のベクトル演算機構とその命令を追加した．

3. 関連研究

実時間 OS を高速化する試みとして，板橋らは，μITRON のシステムコールを高速化する外付けチップを提案している [3]．安堂らは，TOPPERS/ASP3 カーネルと，その上で動作するタスクを高位合成によりフルハードウェア化する手法を提案している [4]．丸山らは，ITRON 仕様実時間 OS をハードウェア化する手法を提案している [5].

RISC プロセッサ上で高級言語の実行を高速化する試みとして，Ungar らは，オブジェクト指向言語である SmallTalk の実行を高速化する RISC プロセッサ SOAR を提案している [6]．SOAR では，SmallTalk において演算実行前に必要な型チェックを高速化するため，compare-and-skip 命令を導入した．

4. 実　　現

4.1 概　　要

我々は，ハードウェア・マルチタスク機構およびエッジ AI 加速命令を搭載した CPU である”松竹 V”を実装した．本 CPU は，東京科学大学吉瀬研究室で開発された RISC-V CPU ”RVCoreP” [7] をベースに，命令などの追加を行った．RVCoreP は RV32I 命令セットを実装した RISC-V 32bit CPU であり，シングルスレッド，インオーダの 5 段パイプラインと gshare 分岐予測を実装している．

4.2 ハードウェア・マルチタスク機構

ハードウェア・マルチタスク機構は，マルチタスクおよびセマフォのハードウェア論理による実現である．タスクやセマフォの操作は，ソフトウェアから表 1 に示す専用命令を実行することで行う．タスクおよびセマフォの最大数は，論理合成時に設定する．

ハードウェア・マルチタスク機構のブロック図を，図 2 に示す．

4.2.1 タ　ス　ク

本 CPU におけるタスクは，スケジューリングの単位である．タスクごとに，独立したプログラム・カウンタと整数レジスタ群を持つ．メモリ空間および分岐

表 1　ハードウェア・マルチタスク命令

命令名	動作	所要サイクル数
task_start rs1, rs2	タスク rs1 をアドレス rs2 から開始する.	1-2
task_terminate rs1	タスク rs1 を停止する.	1-3
task_getid rd	自タスクのタスク ID を取得する.	1
task_read_xreg rd, rs1, rs2	タスク rs1 の整数レジスタ rs2 を読み出す.	1
task_write_xreg rd, rs1, rs2	タスク rs1 の整数レジスタ rd に rs2 の値を書き込む.	1
task_setts rs1, rs2	タスク rs1 のタイムスライスを rs2 サイクルに設定する.	1
task_sleep rs1, rs2	タスク rs1 を rs2 サイクル間スリープさせる.	1-3
task_yield	タスクを切り替える.	1
sem_init rs1, rs2	セマフォ rs1 の値を rs2 に初期化する.	1
sem_signal rs1	セマフォ rs1 に Signal 操作を行う.	1-5
sem_wait rd, rs1, rs2	セマフォ rs1 にタイムアウト時間 rs2 サイクルで Wait 操作を行う. rd が 0 のとき成功，-1 のときタイムアウト.	1-6

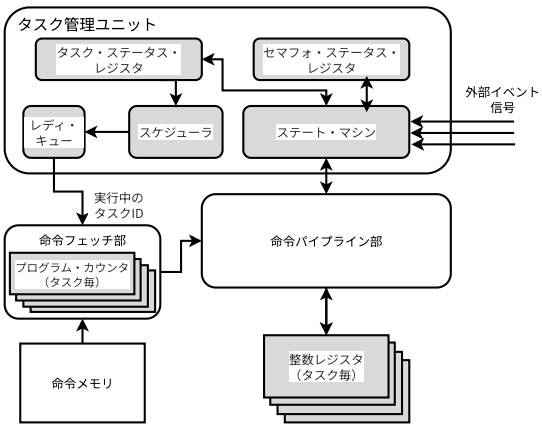

図 2　ハードウェア・マルチタスク機構のブロック図

予測用のテーブルは全タスクで共有する．タスクごとにタイムスライス値をサイクル数単位で設定でき，タスクが与えられたタイムスライスを消費すると，タスク切り替えが発生する．また，タスクは指定したサイクル数だけスリープさせることができる．

4.2.2　セ マ フ ォ

セマフォは，8-bit のカウンティング・セマフォである．タスクはセマフォに対し，Wait 操作および Signal 操作を行える．セマフォごとに，セマフォを待つタスクのキューを持つ．

4.2.3　外部イベント機構

外部イベント機構は，割り込みに代わる機能である．本機構は，セマフォを外部信号との同期に使用することで実現した．タスクは，外部信号イベントと結びつけたセマフォを待つことで，外部イベント発生時に起床される．

4.2.4　スケジューラ

スケジューラは，「実行可能であるがレディ・キューにない」タスクをラウンドロビン方式で選択し，レディ・キューにつなぐ．「実行可能であるがレディ・キューにない」タスクであるかは，タスク・ステータス・レジスタの値より決定する．ラウンドロビン方式でタスクを選択するために，プライオリティ・エンコーダを用いる．

4.2.5　レディ・キュー

レディ・キューは双方向循環連結リストであり，タスク ID が入り，先頭要素と末尾要素の 2 つのレジスタからなる．キューは，タスク・ステータス・レジスタ中に設けた，次要素・前要素の 2 つのリンク情報を用いて実現している．セマフォ待ちタスクのキューも同様のデータ構造であり，先頭要素・末尾要素のレジスタをセマフォ・ステータス・レジスタ中に持つ．

本キューの先頭にあるタスクが実行中タスクであり，このタスクの ID は命令フェッチ部へ出力される．命令フェッチ部は，実行中タスクのプログラム・カウンタを使用して命令フェッチを行う．

タスク切り替え時は，本キューがステート・マシンによりローテートされる．ローテートは 1 サイクルで行えるため，最短で 1 サイクルごとにタスクを切り替えることができる．

4.2.6　ステート・マシン

ステート・マシンは，ハードウェア・マルチタスク命令実行時の処理，タスク切り替え，および外部イベントの発生時の処理を行う．

a)　ハードウェア・マルチタスク命令の処理

命令パイプライン部からの信号により，ハードウェア・マルチタスク命令の発行を検出する．命令に応じて，タスク・ステータス・レジスタやセマフォ・ステー

タス・レジスタの更新，タスク切り替えなどを行う．連結リストの操作には，複数サイクル必要な場合がある．各命令は，長くとも 6 サイクルで完了する．

b） タスク切り替え

以下のいずれかの場合に，ステート・マシンはタスク切り替えを行う．

- 実行中タスクによる，タイムスライス消費・スリープ・セマフォ待ち・自発的なタスク切り替えが発生したとき
- あるタスクが，タイムアウトまたはセマフォ待ちから起床したとき

後者のケースでは，起床したタスクはレディ・キューの先頭に挿入される．すなわち，起床したタスクはすぐに実行が開始される．

c） 外部イベント発生時の処理

外部イベントの発生を検出したとき，外部イベント用のセマフォを待っているタスクがあれば，そのセマフォに Signal 操作を行う．これにより，外部イベントを待っていたタスクは起床させられる．

4.2.7 タスク・ステータス・レジスタ

タスク・ステータス・レジスタは，タスクごとに以下の情報を保持する：

タスクが有効であるか，タイムアウト状態，セマフォ待ち状態，キューイング状態，次要素・前要素タスク ID（連結リスト用）

4.2.8 セマフォ・ステータス・レジスタ

セマフォ・ステータス・レジスタは，セマフォごとに以下の情報を保持する：

カウント値，セマフォ待ちキューが空であるか，セマフォ待ちキューの先頭要素・末尾要素タスク ID

4.2.9 プログラム・カウンタおよび整数レジスタ群

プログラム・カウンタおよび整数レジスタ群は，タスクの数だけ存在する．したがって，タスク切り替え時のレジスタ退避と復元が不要である．

4.2.10 各パイプライン・ステージのタスク ID レジスタ

各パイプライン・ステージに，命令のタスク ID を保持するレジスタを設けた．このタスク ID は，整数レジスタへのアクセス時や，ハードウェア・マルチタスク命令の実行時に，どのタスクの命令であるかを識別するために使用される．また，タスク切り替えの直後は命令パイプライン中に複数タスクの命令が存在しうるが，異なるタスク間でフォワーディングが行われないよう，本レジスタの値を用いて制御を行う．

4.3 多方向ジャンプ命令の設計

4.3.1 概　　要

多方向ジャンプ命令は，動的な高級言語処理系の実行系の実現で頻出する，複数アドレスへのジャンプを高速化するために考案し，実現した．本 CPU に追加した，多方向ジャンプ命令およびレジスタ間転送命令を表 2 に示す．いずれの命令も，1 サイクルで実行できる．本命令の機構は，特許取得済みである(注1)．

4.3.2 レジスタ間接レジスタ指定アドレッシング・モード

本命令のために，「レジスタ間接レジスタ指定アドレッシング・モード」を導入した．オペランドとしてレジスタを指定し，そのレジスタの内容の下位 5 ビットで x0 から x31 のレジスタを選択する．表 2 の表記 `[rs1]` は，本アドレッシング・モードであることを示す．本アドレッシング・モードの実現のため，レジスタ・ファイルの出力をレジスタ・ファイルのアドレス信号に入力するデータ・パスを追加した．

4.3.3 GNU Prolog コードの高速化

図 1 に示したランタイム関数を呼び出す箇所は，以下の流れである．

（1） 引数レジスタ（x10 から x14）へ型ごとのジャンプ先アドレスをセット

（2） `Pl_Switch_On_Term()` 関数を呼び出し，タグの型情報から適切なジャンプ先アドレスを返す

（3） 返り値のアドレスへジャンプ

これを，以下のように書き換えることができ，関数呼び出しや複数回の比較を省略できる．

（1） 引数レジスタ（x10 から x14）へ型ごとのジャンプ先アドレスをセット

（2） タグに対して論理演算および加減算を行い，レジスタ番号と合うように調整

（3） `branch_reg_indirect` 命令の実行

4.4 ベクトル・ユニットの設計

4.4.1 概　　要

ベクトル・ユニットは，以下より構成される．

- ベクトル・レジスタ（1024 ビット幅 ×32 個）
- 8-bit 浮動小数点数パイプライン演算器
- 整数演算器

ベクトル命令は，RISC-V Vector Extension Version 1.0 [8] のサブセットを実装した．以下の命令をサポー

(注1)：特許第 7421850 号，間接アドレス指定方式の条件ジャンプ命令を実行するプロセッサ，プログラム及び方法，竹岡 尚三/木下 喜夫

表 2　多方向ジャンプ命令およびレジスタ間転送命令

命令名	動作
branch_reg_indirect rd, [rs1]	プログラム・カウンタ相対でジャンプを行う．
jump_reg_indirect rd, [rs1], rs2	rs2 をベースアドレスとしてジャンプを行う．
mov_reg_regindirect rd, [rs1]	ソース・オペランドをレジスタ間接レジスタ指定で，レジスタ間転送を行う．
mov_regindirect_reg [rd], rs1	デスティネーション・オペランドをレジスタ間接レジスタ指定で，レジスタ間転送を行う．

トする．

- 8-bit 浮動小数点数（指数 4-bit，仮数 3-bit） 加算・減算・乗算命令
- 8,16,32-bit 整数加算・減算・論理演算・シフト演算命令
- 8,16,32-bit ロード・ストア命令（ストライドロード・ストアを含む）
- Configuration-Setting 命令

なお，8-bit 幅の浮動小数点数演算については，RISC-V 仕様では規定されていない．本実装では，vtype レジスタの vsew フィールドが 0 のときに浮動小数点数演算命令を実行すると，8-bit 幅で演算を行うようにした．

4.4.2　8-bit 浮動小数点数パイプライン演算器

8-bit 浮動小数点数の加算器は 11 段パイプライン，乗算器は 12 段パイプラインである．いずれも，4-way SIMD である．

4.4.3　整数演算器

整数加減算，論理演算およびシフト演算のため，32-bit 幅の整数演算器を持つ．1 要素の演算を 1 サイクルで行える．

4.5　動的クロック切り替え

本 CPU は動作中にクロックを変更することができ，アプリケーションから状況に応じて，高速動作と低消費電力動作を切り替えることができる．専用のレジスタに書き込みを行うことで，外部から入力したクロックの 1〜2^{17} 分周クロック，または PLL クロックの 1〜2^{17} 分周クロックへ切り替えることができる．

5.　評　　価

5.1　評 価 方 法

5.1.1　実チップによる，ベンチマーク，消費電力量およびゲート数の評価

ベンチマーク，消費電力量およびゲート数の評価は，TSMC 90nm プロセスで製造した LSI を使用した．ハードウェア・マルチタスクのタスク数は 4，セマフォ数は 8 に設定した．CPU の動作周波数は，設計上の最大値である 320MHz で評価した．

表 3　CoreMark ベンチマークのスコア

ベンチマーク名	スコア
CoreMark	333.00
CoreMark/MHz	1.04

テープアウト前の検証として，RTL シミュレーションおよびゲートレベル・シミュレーションでプログラムを実行し，期待値との比較を行った．また，一部モジュールについては，オープンソースのフォーマル検証ツールである SymbiYosys を使用してプロパティ検証を行った．

5.1.2　RTL シミュレーションによる，分岐予測ヒット率の評価

分岐予測ヒット率の評価は，RTL シミュレーションにより行った．

5.2　評 価 結 果

5.2.1　ベンチマーク

CoreMark ベンチマークのスコアを，表 3 に示す．C コンパイラは gcc 12.2.0 を使用した．

5.2.2　分岐予測ヒット率

1 個または 2 個のタスクを動作させたときの，分岐回数および分岐予測ヒット率を表 4 に示す．「同アドレスのプログラム」は両タスクが同じ関数を呼び出す場合，「異アドレスのプログラム」は内容が同一の関数を 2 個定義して各タスクから呼び出す場合である．

分岐先アドレスを記憶する Branch Target Buffer は 16 エントリ，パターン履歴を記憶する Pattern History Table は，64 エントリである．

タスク数を 1 個から 2 個にすると，予測ヒット率は，12-16 ポイント低下した．2 個のタスクで異なるアドレスの関数を実行したときは，同じ関数を実行したとに比べ 3-4 ポイント程度ヒット率が低下した．これは，分岐予測テーブルを共有する 2 個のタスクで異なるアドレスに置かれたプログラムを実行することから，予測がミスする確率が高まったと考えられる．

5.2.3　消 費 電 力

LAXER-SoC の消費電力を表 5 に示す．

タスク数を 1 から 4 に増加させると，6mW 程度消

表 4　シングルタスクおよびマルチタスク動作時の分岐予測ヒット率

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

表 5　LAXER-SoC の消費電力

動作状態	消費電力 (mW)		
	1V コア電源	3.3V IO 電源	合計
全タスク停止時	117	10	127
1 タスク動作時	138	10	148
4 タスク動作時	145	9	154
浮動小数点数ベクトル演算時	136	10	146
ROS2 Publisher 動作時	119	10	129

表 6　松竹 V のモジュールごとのゲート数およびその割合

階層名	ゲート数	割合 (%)
松竹 V	428,062	100.00
整数レジスタ	46,511	10.87
タスク管理ユニット	5,499	1.28
ベクトルユニット	347,699	81.23
ベクトルレジスタ	325,345	76.00
8-bit 浮動小数点数演算器	15,478	3.62
整数演算器	1,667	0.39
ALU	1,896	0.44
BTB	508	0.12
PHT	1,542	0.36

費電力が増加した．ベクトル演算実行中でも，消費電力に大きな増加はない．ハードウェア ROS2 ノードを動作させても，消費電力の増加は僅かである．

5.2.4　ゲ ー ト 数

松竹 V のモジュールごとのゲート数と，松竹 V 全体に対する割合を，表 6 に示す．

タスク管理ユニットは，松竹 V 全体の 1.28% である．マルチタスク化にあたって，整数レジスタ以外のゲート数増加は比較的小さいといえる．ベクトルレジスタによるゲート数の増加が大きい．

6. お わ り に

本稿では，エッジ制御向け SoC へ搭載した松竹 V CPU の設計について述べた．LSI を製造し，性能や消費電力，ゲート数などの評価を行った．

今後の課題として，エッジ AI 加速命令の評価が挙げられる．

文　　献

[1] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, “Angel-eye: A complete design flow for mapping cnn onto embedded fpga,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.37, no.1, pp.35–47, 2018.

[2] A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort, “Fp8 quantization: The power of the exponent,” Advances in Neural Information Processing Systems, vol.35, pp.14651–14662, 2022.

[3] 光義板橋, A. Utama, 巧　仲野, 彰睦塩見, 正治今井, “リアルタイム os のハードウェア化とその評価, ” Technical Report 111(1993-ARC-103), 豊橋技術科学大学情報工学系, 豊橋技術科学大学情報工学系, 豊田工業高等専門学校情報工学科, 豊橋技術科学大学情報工学系, 豊橋技術科学大学情報工学系，dec 1993.

[4] 拓也安堂，雄吾石井，菜岐佐石浦，宏之冨山，弘之神原，“Rtos 利用システムの汎用高位合成系を用いたフルハードウェア化, ” Technical Report 3，関西学院大学, 関西学院大学, 関西学院大学, 立命館大学, 京都高度技術研究所，jan 2022.

[5] 丸山修孝，“リアルタイム os のハードウェア化によるマルチコア組込みシステムの高速化, ” ••，pp.••–••，2015.

[6] D. Ungar, R. Blau, P. Foley, D. Samples, and D. Patterson, “Architecture of soar: Smalltalk on a risc,” SIGARCH Comput. Archit. News, vol.12, no.3, pp.188–197, Jan. 1984. https://doi.org/10.1145/773453.808182

[7] H. MIYAZAKI, T. KANAMORI, M.A. ISLAM, and K. KISE, “Rvcorep: An optimized risc-v soft processor of five-stage pipelining,” IEICE Transactions on Information and Systems, vol.E103.D, no.12, pp.2494–2503, 2020.

[8] “Risc-v ”v” vector extension,” 2021. https://github.com/riscvarchive/riscv-v-spec/releases/download/v1.0/riscv-v-spec-1.0.pdf

（xxxx 年 xx 月 xx 日受付）

松永　大輝

情報処理学会 正会員．2021 年 株式会社アックス 入社．

竹岡　尚三

ACM 正会員．IEEE 正会員．一般社団法人組込みシステム技術協会 理事，技術本部長．PC クラスタコンソーシアム 理事．DEOS 協会 理事，事務局長．OSS コンソーシアム 副会長，理事．株式会社アックス 代表取締役社長．

Abstract In this study, we developed an enhanced RISC-V CPU for use in edge control SoCs. This CPU realizes the functions of a real-time OS such as a task scheduler and a semaphore, which are usually implemented by software, by hardware. As an alternative to the interrupt mechanism, it has an external event notification mechanism using hardware semaphores. Chips used for edge control in industrial edge devices and robots tend to have small amounts of memory due to the need for low power consumption and cost. By implementing functions equivalent to real-time OSes in hardware, the memory footprint can be reduced and overhead such as task switching can also be reduced. Because it is implemented in hardware, the OS-equivalent functions are also robust. In addition, to speed up edge AI applications, 8-bit floating-point number vector operation instructions and multi-directional jump instructions have been added. The logic with this CPU was manufactured by TSMC 90nm process, and it was completed as a SoC of 9.2 million transistors.

Key words CPU, RTOS, RISC-V, ASIC, Prolog