電気学会論文誌 D（産業応用部門誌）
IEEJ Transactions on Industry Applications
Vol.143 No.2 pp.94–100　DOI: 10.1541/ieejias.143.94

論　文

高位合成可能なソフトプロセッサにおける専用命令実装手法の評価

非会員　岩原　和輝*　正　員　市川　周一*a)　非会員　藤枝　直輝**

Evaluation of Special Instruction Implementations in Soft Processors for High-level Synthesis

Kazuki Iwahara*, Non-member, Shuichi Ichikawa*a), Member, Naoki Fujieda**, Non-member

（2022年3月16日受付，2022年5月14日再受付）

It is important to protect the intellectual property (IP) of embedded software, which contains precious know-hows and trade secrets. As hardware is more difficult to analyze than software, it is possible to protect IP by implementing it in hardware. It is particularly easy to port software to hardware when both the software and hardware are written in the same language. Some recent soft processors have been written in a high-level programming language (e.g., C), which can be synthesized by high-level synthesis (HLS) tools. Sakamoto et al. (2018) implemented a function of a software as a special instruction of a MIPS-based soft processor. They also pointed out that passing arguments and values involves various problems. Iwamoto et al. (2019) and Masanobu et al. (2020) enhanced Sakamoto's method; however, their evaluation results were imperfect and problematic. This study resolves many of these problems and reports more evaluation results than previous works. Detailed evaluation results suggest that the hardware implementation of a function is possible and that the overhead in latency and hardware cost is modest.

キーワード：高位合成（HLS），専用命令，ソフトプロセッサ，CHStone
Keywords: High-level Synthesis (HLS), custom instruction, soft processor, CHStone

1. はじめに

組込みソフトウェアの知的財産保護は重要な課題である。組込みソフトウェアには多くの技術やノウハウが含まれており，盗用されれば所有者にとって大きな損失となる。特にソフトウェアは複製や解析が容易であるため，剽窃や改変の被害を受けやすいという特徴を持つ。

一般にハードウェアはソフトウェアより解析が難しいため，ソフトウェアの一部を論理回路として実装することにより，アルゴリズムやデータを秘匿して知的財産を保護することができる[(1)~(3)]。組込みシステムは利用目的が明確に決まっているため，搭載するソフトウェアも出荷時に定まっていることが多い。製品毎にハードウェアをカスタマイズすることや，特定応用に対応する専用回路を搭載することも広く行われている。そのため組込みソフトウェアの一部を専用回路化することは，極めて自然な発想である。

こうして作成した専用回路を使用する一つの方法は，周辺回路として既製プロセッサに付加することである。周辺回路は実装や利用が容易だが，一般にペリフェラルバスのインタフェースを介した接続となり，その実装コストが問題となる。組込み OS 側ではデバイスドライバも必要である。ペリフェラルバス経由のデータ転送に大きな遅延が発生し，性能面のボトルネックになる場合もある[(4)]。

もう一つの方法は，専用回路をプロセッサ内部に実装し，ソフトウェアから特殊命令として利用する方法である。応用志向の命令を拡張したプロセッサ（ASIP；Application-Specific Instruction-set Processor）は，システムの性能向上を目的として広く用いられている[(5)]。従って知財保護を目的とした ASIP を作成することも技術的に可能である。しかし ASIP の開発には時間と労力が必要で，必要コストを上回る利益がないと開発が正当化されない。LSI が量産前提の技術である以上，信号処理や暗号化のように広く使わ

a) Correspondence to: Shuichi Ichikawa. E-mail: ichikawa@ieee.org
* 豊橋技術科学大学 電気・電子情報工学専攻
〒441-8580　愛知県豊橋市天伯町雲雀ヶ丘 1-1
Department Electrical and Electronic Information Engineering, Toyohashi University of Technology
1-1, Hibarigaoka, Tampaku-cho, Toyohashi, Aichi 441-8580, Japan
** 愛知工業大学工学部 電気学科
〒441-8580　愛知県豊橋市天伯町雲雀ヶ丘 1-1
Department Electrical and Electronics Engineering, Faculty of Engineering, Aichi Institute of Technology
1-1, Hibarigaoka, Tampaku-cho, Toyohashi, Aichi 441-8580, Japan

れる特殊命令は正当化できるが，特定の応用のみが使用する特殊命令は正当化されにくい。

近年，FPGA（Field Programmable Gate Array）技術の進歩により，システム LSI の単品生産が容易になった。さらに CAD 技術の進歩によって，開発コストも大きく削減された。プロセッサもハードウェア記述言語（HDL；Hardware Description Language）で記述され，論理合成技術によって自動生成できる。HDL で記述されたプロセッサ（ソフトプロセッサ）からは，使用目的に応じて低コスト～高性能まで様々なプロセッサを生成可能である。こうして，用途の限定された ASIP も開発・製造できるようになった。

最後に残った問題は，応用に特化した専用回路を開発する方法である。論理合成は HDL から論理回路を生成するが，応用ソフトウェアはプログラミング言語で書かれているため，回路化するにはプログラムを HDL に変換する必要があった。しかしこれも近年の高位合成（HLS；High-Level Synthesis）技術の進歩により解決されつつある。HLS はプログラミング言語（例えば C 言語）から論理回路を生成できるため，ハードウェアとソフトウェアの境界が柔軟になった（HW/SW 協調設計）。

本研究では，知財保護を目的として，応用プログラムの一部をソフトプロセッサの特殊命令として実現する方法について検討する。さらに特殊命令の実装によりプロセッサのコストと性能がどのような影響を受けるか定量的に評価する。

なお本稿は，著者らによる研究会発表[6]に加筆修正を施したものである。

2. 関連研究

高位合成可能なソフトプロセッサは，様々な目的で提案されている。Skalicky ら[7]は，C 言語で記述された命令セットシミュレータを高位合成することによりソフトプロセッサを生成することを提案した。Rokicki ら[8]は，C/C++で記述された RISC-V アーキテクチャのソフトプロセッサ Comet を提案した。Comet は 5 段パイプラインを持ち，明示的にパイプライン構造が記述されている。Mantovani ら[9]は，SystemC 言語で記述された 32 ビット RISC-V プロセッサ HL5 を提案し，商用 HLS（Stratus HLS）で合成して Xilinx Zynq SoC FPGA で実装評価した。

坂本ら[10]は，応用プログラムの関数をソフトプロセッサの特殊命令として実装することを提案した（Fig. 1）。ソフトプロセッサと応用が同じ言語（C 言語）で記述されていれば，応用の一部をプロセッサ内に移動することは比較的容易である。Fig. 1(a) は移動前，(b) は移動後を表している。Fig. 1(b) では，隠蔽したい関数 Function() の内容（Func. code）をソフトプロセッサ内に移動し，特殊命令として実現している。応用プログラムの関数 Function() 内では，Func. code の代わりに対応する特殊命令（Sp. Inst.）を埋め込んで，特殊命令を実行することにより処理を完遂する。

このアプローチでは，回路化する関数毎に一つの専用命令を作成すればよく，複数の応用（関数）を同時に回路化することが容易である。専用命令は asm 文で埋め込むことができるので専用のコンパイラやライブラリが不要であり，専用命令を使わない応用や OS に影響を及ぼさない。その一方で，引数や値の受け渡し方法には工夫が必要である。専用命令のオペランドは命令形式で決まっているが，ソフトウェアの関数は引数の型も個数も自由であるため，専用命令化する際に何らかの工夫が必要になる。(3 章参照)。

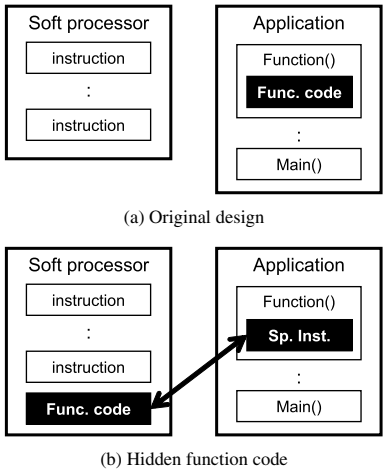

Fig. 1. Concept of special instruction.

坂本ら[10]は，MIPS アーキテクチャのソフトプロセッサ Spim-like を自作し，Skalicky ら[7]の手法をベースにして専用命令を実装した。専用命令化する対象アプリケーションとしては CHStone ベンチマーク[11]のうち 3 種を使用し，引数の受け渡し方法についても検討した。しかしながら，専用命令化する関数の選択方法がアドホックであり，評価対象のアプリケーションが 3 種類と少ないことが問題であった。

そこで岩本ら[12]は，坂本ら[10]の手法を再検討し改良した。関数の選択方法に 2 つの基準（実行時間とコードサイズ）を設け，対象アプリケーションも CHStone のうち 11 種類へ拡張した。さらに引数の受渡し方法 3 種についても，実装評価して比較した。しかし 39 種類の実装パターンのうち 8 種類で不具合が発生し，評価対象から除外している。不具合の内容は，高位合成時のエラーと，シミュレーション時のエラーである。

正信ら[13][14]は，岩本ら[12]の評価結果を詳細に再検討し，実際の不具合は岩本らの報告よりも多いことを指摘した。再評価の結果では，39 種のうち 23 種の実装パターンで不具合が生じていた。これでは不具合が多すぎて，定量的評価として信頼性に欠ける。

新たに発見された不具合は，高位合成やシミュレーションは正常に終了するが，回路から出力される値が正しくないという問題である。CHStone は高位合成用のベンチマークであるためテストベッド機能があり，計算された値が期待値と一致するか確認している。各テストでは複数の値を計算し，期待値と不一致を起こした回数を値として返す。各テストが正常終了すれば，返り値は全て 0 になるはずで

ある。

本研究では正信らの指摘した不具合を調査し，その原因を明らかにする。さらに不具合を解消して再評価を行い，専用命令を付加したプロセッサの性能と回路規模について，定量的評価を行うことを目的とする。

3. 専用命令実装手法

本研究では，先行研究[10][12][13]と同じくMIPS32-R1アーキテクチャのソフトプロセッサ（Spim-like）を評価基盤として用いる。さらにSpim-likeに各アプリケーションの一部を専用命令として拡張し，専用ソフトプロセッサ（Spim-like_sp）を作成する。Fig. 1(a)のソフトプロセッサがSpim-likeであり，(b)のソフトプロセッサがSpim-like_spに対応する。

2章でも述べた通り，アプリケーションの関数を専用命令化するためには，ソフトウェアと専用命令の間の引数・値の受け渡し方法が問題になる。この実装方法に応じて，作成するSpim-like_spの論理規模やアプリケーションの実行時間が影響を受けるため，定量的な評価による比較が必要である。

本研究では，先行研究[13][14]に従い，以下の4通りの設計で評価を行った。

〈3･1〉 値渡し：sp　一般にMIPS命令セットでは，一定の条件を満たす場合，レジスタを使って関数の引数と値を受け渡す。本研究で用いるCHStoneは固定小数点演算を行うため，引数には汎用レジスタ$a0～$a3，値には$v0と$v1が用いられる†。

本研究でも，専用命令で汎用レジスタを用いた受け渡しが可能であれば，レジスタを用いた値渡しを行う。この実現方法を，以下**sp**と呼ぶ。

〈3･2〉 参照渡し：ptr　引数の数が多すぎる場合やサイズが大きすぎる場合は，引数や値をレジスタで受け渡すことができない。その場合，メモリを介して引数や値を受け渡す。コンパイラなどは通例スタック上に領域を確保する。こうした場合，引数を格納した領域のアドレスを専用命令に渡す。

具体的には，引数領域の実効アドレスをレジスタに作成して，そのレジスタの番号を専用命令のrsフィールドとして渡す。この実装を，以下**ptr**と呼ぶ。この実装では，専用回路内にメモリアクセスの論理が付加され，専用命令からメモリを直接操作する。

〈3･3〉 スクラッチパッド：spm_s　専用命令から直接メインメモリを操作すると，メモリアクセスが処理のボトルネックになる可能性がある。そこでプロセッサ内に引数用メモリ（SPM；scratch pad memory）を用意し，専用命令はメインメモリの代わりにSPMを操作するようにする。ソフト側からSPMを読み書きするために，SPMへのロード・ストア命令を別に実装する。

関数は，引数全てをSPMにストアしてから，専用命令を呼び出す。専用命令はSPM上の値を用いて処理を行い，値をSPMに書き込む。専用命令実行後，関数はSPMから値をロードしてリターンする。この実装方法を以下**spm_s**と呼ぶ。

複数の専用命令を実装した場合も，SPMは1つであり，共用される。専用命令の前後で引数・値を受け渡すため，関数（ソフトウェア）側にSPMへのロード命令やストア命令が必要である。

〈3･4〉 スクラッチパッド：spm_h　SPMへのロード・ストアをソフトウェアで行う代わりに，その操作をハードウェアで行う方法を，以下**spm_h**と呼ぶ。この設計では引数はアドレスで渡され，専用命令側でSPMにロード・ストアする。そのため関数（ソフトウェア）側のコードはptrと同じになる。専用命令を複数実装した場合，SPMは各命令に固有のメモリとなる。必要なハードウェア資源は大きくなるが，メインメモリへのアクセスは処理の最初と最後だけなので，性能面で有利になると考えられる。

4. 実装と評価

〈4･1〉 評価環境　本研究で使用するソフトウェア開発環境とハードウェア開発環境を，それぞれTable 1とTable 2にまとめる。評価対象には，先行研究[10][12][13]と同じCHStoneベンチマーク[11]を用いる。CHStoneはTable 3に示す12種のアプリケーションからなる。そのうちMIPSはmain関数だけで構成されており，これを専用命令として秘匿すると，アプリケーション全体がハードウェア化されることになるので，本研究の趣旨から外れる。そのため先行研究[12][13]と同様，本研究でもMIPSを除く11種のアプリケーションについて評価を行う。

〈4･2〉 専用命令化の対象　アプリケーションが複数の関数からなる場合，どの関数を専用回路化するか選択肢が生じる。まず第一に，使用するHLSの制約に抵触しない関数を選ぶ必要がある。例えばVivado HLSでは再帰関数は合成できず，ポインタの型変換も制限される。CHStoneは高位合成用ベンチマークなので再帰関数は含まれていな

Table 1. Application development environment.

VM Software	Oracle VM VirtualBox
Guest OS	Ubuntu 18.04
Compiler	Mipsel-linux-elf_gcc ver5.4
Compiler Option	-EL -mips32 -static -nostdlib -fno-builtin -msoft-float
Architecture	MIPS32 Release 1
Benchmark Software	CHStone v1.11

Table 2. Hardware development environment.

HLS Tool	Vivado HLS 2020.1
Logic Synthesis Tool	Vivado 2020.1
Target Device	Artrix-7 xc7a100tcsg324-1

† CHStoneは高位合成用のベンチマークなので，高位合成でサポートされない浮動小数点演算は含まれていない。アプリケーションにDFADDなど浮動小数点系の演算が含まれるが，これは固定小数点演算を用いて浮動小数点演算を実現する（いわゆる）soft-float処理を記述したものである。

Table 3. CHStone benchmarks [11].

Application	Description
ADPCM	Adaptive differential pulse code modulation decoder and encoder
AES	Advanced encryption standard
BLOWFISH	A data encryption standard
DFADD	Double-precision floating-point addition
DFDIV	Double-precision floating-point division
DFMUL	Double-precision floating-point multiplication
DFSIN	Double-precision floating-point sine function
GSM	Linear predictive coding analysis of global system for mobile communications
JPEG	JPEG image decompression
MOTION	Motion vector decoding for the MPEG-2
MIPS	Simplified MIPS processor
SHA	Secure hash algorithm

Table 4. Functions implemented as a special instruction.

Application	Selection criteria	Function
ADPCM	exec. time	upzero
	code size	reset
AES	exec. time	AddRoundKey_InversMixColumn
	code size	ByteSub_ShiftRow
BLOWFISH	time/size	BF_encrypt
DFADD	time/size	propagateFloat64NaN
DFDIV	time/size	mul64To128
DFMUL	time/size	mul64To128
DFSIN	time/size	mul64To128
GSM	exec. time	gsm_mult
	code size	gsm_norm
JPEG	exec. time	YuvToRgb
	code size	ChenIDct
MOTION	time/size	Flush_Buffer
SHA	time/size	sha_transform

いが，一般に HLS が扱える構文には制約があり，その制約は使用する HLS の製品毎に異なっている。さらに，専用命令化する関数は，その関数から呼び出される関数が他の関数で使用されていないものであることが望ましい[12]。

こうした制約を満たしたうえで，秘匿する関数は作者の意志（主観）で選ぶことができる。しかし本研究では既製のベンチマークを用いて客観的評価を行うため，以下の 2 つの基準（Selection criteria）で関数を選択した。

実行時間（execution time）。 動的な選択基準。実行時間の長い処理が重要なので秘匿したいという考え方。

コードサイズ（code size）。 静的な選択基準。メモリ上で大きな領域を占める処理を秘匿したいという考え方。

各アプリケーションのオブジェクトファイルと実行時間プロファイルから，上記の観点で秘匿対象の関数を選んだ。選択結果を Table 4 にまとめる。幾つかのアプリケーションでは，どちらの基準でも同じ関数が選択された。

〈4･3〉 不具合の検証 本研究では，正信ら[13][14]の再現実験を行い，動作異常について個別に原因を究明して修正した。まず Vivado HLS 上で Spim-like_sp のアプリケーションバイナリをソフトウェアとして実行し，C シミュレーションを行って各テストの戻り値をチェックした。Table 5 に各テストの実行結果をまとめる。

Table 5. Self-check results of Spim-like_sp.

Application	Selection criteria	Impl.	Self-checking value		Remark
			Original	Current	
ADPCM	execution time	ptr	0	0	-
		spm_s	-	0	-
		spm_h	0	0	-
	code size	ptr	0	0	-
		spm_s	96	e	NG
		spm_h	0	0	-
AES	execution time	ptr	0	0	-
		spm_s	10	0	-
		spm_h	10	0	-
	code size	ptr	0	0	-
		spm_s	1f	0	-
		spm_h	0	0	-
BLOWFISH	time/size	ptr	0	0	-
		spm_s	1441	0	-
		spm_h	1440	0	-
DFADD	time/size	sp	c	0	-
DFDIV	time/size	ptr	4	0	-
		spm_s	4	4	NG
		spm_h	4	0	-
DFMUL	time/size	ptr	0	0	-
		spm_s	8	0	-
		spm_h	8	0	-
DFSIN	time/size	ptr	0	0	-
		spm_s	23	23	NG
		spm_h	23	0	-
GSM	execution time	sp	0	0	-
	code size	sp	0	0	-
JPEG	execution time	ptr	3cbd	3cbd	NG
		spm_s	3c96	3c96	NG
		spm_h	3cbd	3cbd	NG
	code size	ptr	-	-	Csim.error
		spm_s	-	-	Csim.error
		spm_h	-	-	Csim.error
MOTION	time/size	ptr	0	0	-
		spm_s	2	0	-
		spm_h	0	0	-
SHA	time/size	ptr	0	0	-
		spm_s	5	0	-
		spm_h	0	0	-

Table 5 に示す 11 種のアプリケーションについて，Table 4 に示した関数を専用命令化した。その際に引数を渡す方法は複数考えられるため，Impl. 列に示した方法（3 章）を実装した。実行結果（Self-checking value）は各関数の戻り値であり，2 章でも説明した通り 0 が正常終了である。Original 列は修正前（正信[13]の実装），Current 列は修正後の値である。

JPEG については，実行時間・コードサイズのいずれの基準で選んだ関数も，正しく実行することができなかった。関数 YuvToRgb の専用命令化については，いずれの設計でも多数の不一致が発生している。関数 ChenIDct の専用命令化では，シミュレーションの段階でエラーとなり，原因究明すら困難な状況である。ADPCM，DFDIV，DFSIN の spm_s については，現在まで不一致を解消できていない。

Table 6. Detailed evaluation results of Spim-like_sp.

App	Selection criteria	Impl.	Latency[cycles]	SLICE	LUT	FF	DSP	BRAM
ADPCM	execution time	ptr	922027	1029	2986	2013	13	2
		spm_s	1459475	904	2873	1774	13	2
		spm_h	931275	1008	2989	1973	13	4
	code size	ptr	1141912	1381	3360	2620	10	2
		spm_h	1142122	1509	4133	2684	10	2
AES	execution time	ptr	274584	1083	3187	2442	11	3
		spm_s	551721	1001	3075	2232	11	6
		spm_h	284196	1159	3288	2496	10	10
	code size	ptr	489321	1545	4028	2926	10	3
		spm_s	509001	993	3005	1975	10	5
		spm_h	490651	990	3002	2039	10	5
BLOWFISH	time/size	ptr	2813380	2925	7780	6066	10	2
		spm_s	39678554	2341	6613	5800	10	6
		spm_h	7697621	2764	6186	6932	10	4
DFADD	time/size	sp	106993	911	2787	1736	10	2
DFDIV	time/size	ptr	302420	994	2984	2076	26	2
		spm_h	302480	1027	3008	2034	26	2
DFMUL	time/size	ptr	44311	1025	3123	1942	26	2
		spm_s	47535	1040	3055	1985	26	4
		spm_h	44367	995	3043	2066	26	2
DFSIN	time/size	ptr	8518397	1025	3123	1942	26	2
		spm_h	8522555	995	3043	2066	26	2
GSM	execution time	sp	260206	831	2586	1684	11	2
	code size	sp	276032	855	2662	1648	10	4
MOTION	time/size	spm_h	23480	1281	3781	2787	10	4
SHA	time/size	ptr	1191326	1198	3694	3019	10	4
		spm_s	1400524	1166	3747	2850	10	6
		spm_h	1219082	1263	3903	3033	10	6

時間の関係上，本研究では JPEG の評価は諦め，今後の課題とした。以下の評価では CHStone のうち MIPS と JPEG を除く 10 種について評価結果を示す。10 種のアプリケーションについて，33 種の実装パターンのうち 30 種で正常動作が確認された。不一致が残った 3 種はいずれも spm_s であるが，原因については調査中である。

〈4·4〉 評価結果 C シミュレーションで正常動作が確認された 30 種の実装（Table 5）について，実行時間とハードウェア資源量を評価した。評価結果の詳細を Table 6 にまとめる。Fig. 2 と Fig. 3 は，Table 6 から実行サイクル数と slice 使用量を取り出し，Spim-like（専用命令なし）の値を 1 として相対値で図示したものである。また Fig. 4 は slice 使用量と実行サイクル数の積（AT 積）を示したもので，AT 積はコスト／性能を表す評価指標である†。

評価では，まず Spim-like_sp を Vivado HLS で高位合成し，Spim-like_sp 用のアプリケーションを用いた協調シミュレーションを行って，実行サイクル数（Latency）を測定する。実行時間はサイクル時間とサイクル数の積であるが，本実験では全ての実装パターンでターゲット周波数を 100 MHz に統一し，サイクル時間は同じ（10 ns）であると仮定した。そのため実行サイクル数を実行時間の評価指標としている。各設計固有の性能チューニングは今後の課題とする。

ハードウェア資源量は，高位合成した Spim-like_sp の HDL

† 基本的に，コストは slice 使用量に比例し，性能は実行サイクル数に反比例するため。

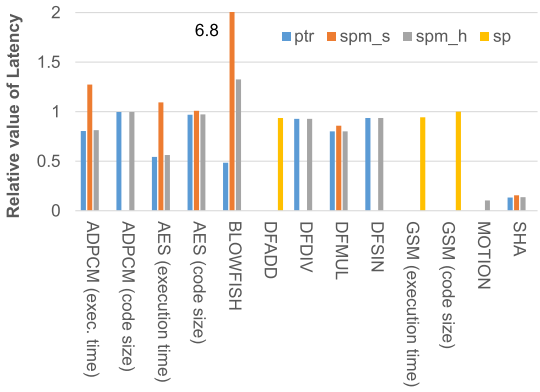

Fig. 2. Latency.

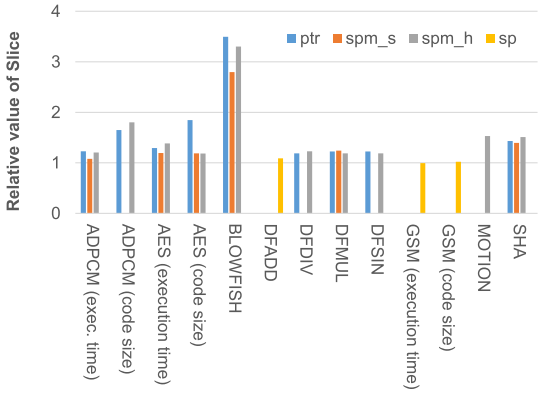

Fig. 3. Hardware resource.

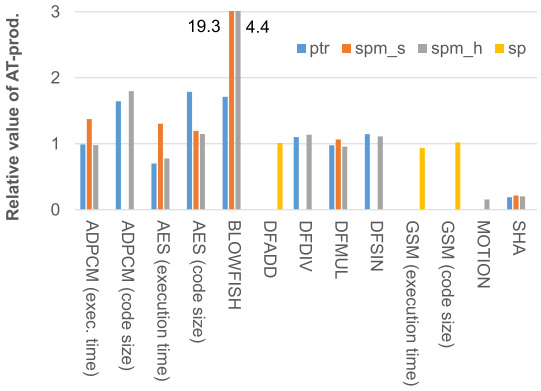

Fig. 4. Area-Time product (AT product).

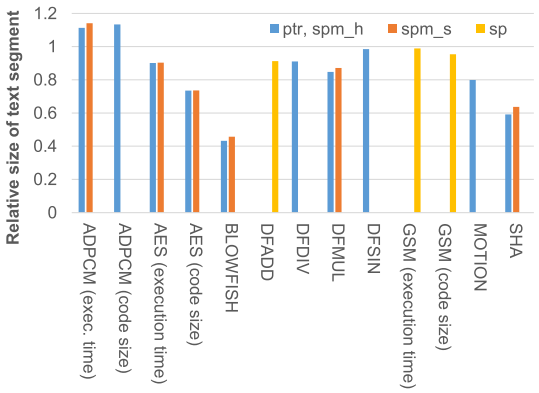

Fig. 5. Size of text segment.

記述を Vivado で論理合成して求めた。ただし MOTION の ptr と spm_s 実装については，合成時にエラーが発生したため，論理規模の見積もりをすることができなかった。そのため本章の評価では上記 2 種を除いた 28 種の実装パターンについて評価結果を示している。FPGA への実装では，資源量は slice，DSP，BRAM などの内訳で示される。本研究で使用した XC7A100T には，15850 個の slice が搭載され，各 slice には 4 つの LUT と 8 つの FF が含まれている[15]。さらに XC7A100T には 240 個の DSP（25×18 乗算器）と 270 個の BRAM（18 Kbit RAM）が搭載されている。

MOTION（ptr）の高位合成エラーは先行研究[13]でも報告されていたものである。エラーは専用回路化した関数内における変数型のキャストであり，Vivado HLS で異なる変数型のキャストがサポートされていないことが原因であると考えられる。こうした問題はソースコードの書き換えで対応可能かも知れないが，それは特定の HLS 製品に固有の変更となる。本研究では，客観的評価を行うために公開されたベンチマークプログラムを使用しているという事情もあり，ソースコードの書き換えは最小限に留めている。単純で機械的な書き換えで対応可能なものは対処したが，大きな変更は敢えて避けて評価対象から外した。

AES（code size）では，ptr に比べて spm_s, spm_h の slice 使用量が小さい（Fig. 3）。直感に反するようだが，spm_s と spm_h では SPM が BRAM で実装されているため，BRAM 量が増えている（Table 6）。BRAM が増える代わり，ptr より FF の使用量が少なくなり，結果として slice 使用量も減ったと考えられる。

BLOWFISH では 3 種の実装全てで論理規模が大きく増大している。これは専用回路化した関数が大きなハードウェア量を要求するためで，ある意味，当然の帰結である。専用回路化により ptr では実行サイクル数が半減しているが，spm_s ではサイクル数が 6.7 倍となっている。BLOWFISH で回路化される関数 BF_encrypt は，ベンチマークのテストパタンで 1000 回以上呼び出され，関数の実行前後にメモリと SPM の間で 2000 個以上のデータがコピーされる。特に spm_s では SPM へのロード・ストア命令のオーバヘッドが大きくなり，実行サイクル数が著しく増加していると考えられる。このような関数でも秘匿化することは可能だが，性能面のオーバヘッドが大きくなることが示された。

BLOWFISH を除けば，多くの実装パターンで論理規模と実行サイクル数への影響は軽微であり，秘匿化のメリットを考えて受入可能であると思われる。ただし BLOWFISH の例からわかる通り，秘匿する関数の選択次第で，論理規模・実行サイクル数の双方で大きな影響が出る可能性があることも分かった。

最後に，関数の専用命令化によるコードサイズの変化を Fig. 5 に示す。コードサイズには，readelf コマンドにより表示される text セグメントのサイズを採用した。関数本体を専用命令化すると，コードサイズは減少することが期待される。しかし引数のアドレスを渡す際（ptr，spm_h），メモリ上にアドレスのリストを準備し，さらに専用命令を呼び出すというオーバヘッドが生じる。ソフトウェアで SPM を使う場合（spm_s），専用命令に加えて SPM へのロード／ストア命令によるオーバヘッドが発生する。

Fig. 5 では，ADPCM においてコードサイズが 10%ほど増加している。これは，ADPCM で専用回路化された関数（upzero, reset）が非常に小さいため，専用命令化によるコードサイズ縮小効果よりオーバヘッドが大きくなったためである。逆に BLOWFISH では 6 割近く削減され，隠蔽効果が大きくなったことが分かる。平均的にはコードサイズが削減され，処理の隠蔽が実現できたといえる。

5. おわりに

本研究では，知財保護を目的として，応用プログラムの一部をソフトプロセッサの特殊命令として実現する方法について検討した。先行研究[12][13]では多くの不具合が残っておりデータの信頼性が不足していたが，本研究では不具合の多くを解決し，データの信頼性を高めることができた。さらに特殊命令の実装によりプロセッサのコストと性能が受ける影響を定量的に評価した結果，本研究の範囲内ではコストと性能への影響は軽微であることが確認できた。

本研究の結果から，高位合成により知的財産を専用回路化して秘匿することが可能であることを示すことができた。専用回路化の手法自体は秘匿化に限定されるものではなく，

性能や消費電力の改善を目的とした専用命令化にも応用可能な知見を示すことができた。

現状の HLS には製品固有の制約が多く，ソフトウェアの専用回路化には多くの制約が課されることが判明した。しかしこうした制約は，今後の CAD 技術の進歩により解決されていくと期待される。本研究で提案した手法により，組込みシステムにおいて専用命令を持つソフトプロセッサの利用が容易になると期待される。

また本研究では，実行時間の評価をサイクル数で行い，主記憶と SPM のアクセス時間を区別していない。このため 4 章の評価でも，ptr に対して spm_s と spm_h の優位性が明確になっていない。FPGA 上の BRAM で主記憶を実現する場合は上記の仮定が成立するが，現実的な組込みシステムでプロセッサ外部に主記憶が存在する場合，チップ内の SPM とチップ外の主記憶は 10 倍以上アクセス時間が異なると考えられる。このような設計に関する性能比較については，今後の課題とする。

謝　辞

本研究の初期段階に貢献のあった，坂本譲二，岩本凌大，正信健人の三氏に感謝いたします。なお本研究の一部は JSPS 科研費 20K11733 の支援によるものである。

文　献

（1） S. Ichikawa, M. Akinaka, H. Hata, R. Ikeda, and H. Yamamoto: “An FPGA implementation of hard-wired sequence control system based on PLC software”, *IEEJ Trans. EEE*, Vol.6, No.4, pp.367–375 (2011)

（2） Y. Matsuoka, N. Fujieda, and S. Ichikawa: “Evaluation of hardware obfuscation techniques using obfuscation tool oLLVM”, *IEEJ Trans. IA*, Vol.139, No.2, pp.111–118 (2019) (in Japanese)
松岡佑海・藤枝直輝・市川周一：「難読化ツール oLLVM を用いたハードウェア難読化手法の評価」, 電学論 D, Vol.139, No.2, pp.111–118 (2019)

（3） S. Yamada, S. Ichikawa, and N. Fujieda: “Implementation of Obfuscated Control Logic Circuit with LegUp and oLLVM”, *IEEJ Trans. EIS*, Vol.139, No.9, pp.952–957 (2019) (in Japanese)
山田翔太郎・市川周一・藤枝直輝：「LegUp と oLLVM による難読化制御論理回路の実装」, 電学論 C, Vol.139, No.9, pp.952–957 (2019)

（4） N. Fujieda, S. Ichikawa, Y. Ishigaki, and T. Tanaka: “Evaluation of the hard-wired sequence control system generated by high-level synthesis”, Proc. 26th IEEE Intl. Symp. Industrial Electronics (ISIE 2017), pp.1261–1267 (2017)

（5） C. Galuzzi and K. Bertels: “The Instruction-Set Extension Problem: A Survey”, *ACM Trans. Reconfigurable Technology and Systems*, Vol.4, No.2, pp.18:1–18:28 (2011)

（6） K. Iwahara, S. Ichikawa, and N. Fujieda: “Implementation method of dedicated instructions in soft-processors for high-level synthesis”, The Papers of Technical Meeting on Innovative Industrial System, IEE Japan, IIS-22-020 (2022) (in Japanese)
岩原和輝・市川周一・藤枝直輝：「高位合成可能なソフトプロセッサにおける専用命令実装手法の検討」, 電学次世代産業システム研, IIS-22-020 (2022)

（7） S. Skalicky, et al.: “Designing customized ISA processors using high level synthesis”, Proc. Intl. Conf. ReConFigurable Computing and FPGAs (ReConFig 2015), pp.1–6 (2015)

（8） S. Rokicki, D. Pala, J. Paturel, and O. Sentieys: “What You Simulate Is What You Synthesize: Designing a Processor Core from C++ Specifications”, Proc. IEEE/ACM Computer-Aided Design (ICCAD 2019), pp.1–8 (2019)

（9） P. Mantovani, R. Margelli, D. Giri, and L.P. Carloni: “HL5: A 32-bit RISC-V Processor Designed with High-Level Synthesis”, Proc. IEEE Custom Integrated Circuits Conference (CICC 2020), pp.1–8 (2020)

（10） 坂本譲二・藤枝直輝・市川周一：「高位合成で生成されたプロセッサに対する特殊命令実装の検討」, 2nd cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG 2018), Session 5B (2018)

（11） Y. Hara, H. Tomiyama, S. Honda, and H. Takada: “Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis”, Journal of Information Processing, Vol.17, pp.242–254 (2009)

（12） R. Iwamoto, N. Fujieda, S. Ichikawa, and J. Sakamoto: “Preliminary evaluation of special instruction implementation methods by high level synthesis”, IEICE Tech. Rep. RECONF2018-62, Vol.118, No.432, pp.101–106 (2019) (in Japanese)
岩本凌大・藤枝直輝・市川周一・坂本譲二：「高位合成による専用命令実装手法の予備的評価」, 信学会技報 RECONF2018-62, Vol.118, No.432, pp.101–106 (2019)

（13） K. Masanobu: “An investignaion of processor implementation methods by high level synthesis”, Master’s thesis, Graduate Program of Electrical and Electronic Information Engineering, Toyohashi University of Technology (2020) (in Japanese)
正信健人：「高位合成によるプロセッサ実装手法の検討」, 修士論文, 豊橋技術科学大学電気・電子情報工学専攻 (2020)

（14） K. Iwahara, K. Masanobu, and S. Ichikawa: “The HLS implementation of special instructions: revisited”, 20th Forum on Information Technology (FIT2021), C-015 (2021) (in Japanese)
岩原和輝・正信健人・市川周一：「高位合成による専用命令実装手法の再検討」, 第 20 回情報科学技術フォーラム (FIT2021), C-015 (2021)

（15） Xilinx: “7 Series FPGAs Data Sheet: Overview”, DS180 (v2.6.1) (2020)

岩　原　和　輝（非会員） 2020 年豊橋技術科学大学電気・電子情報工学課程卒業。同年，同大学大学院電気・電子情報工学専攻博士前期課程入学。2022 年 3 月，同大学大学院電気・電子情報工学専攻博士前期課程修了。

市　川　周　一（正員） 1985 年東京大学理学部卒業。1987 年同大学大学院理学系研究科修士課程修了。1987 年新技術事業団，1991 年三菱電機（株），1994 年名古屋大学工学部助手。1997 年豊橋技術科学大学工学部講師。同助教授，准教授を経て，2011 年沼津工業高等専門学校制御情報工学科教授。2012 年より豊橋技術科学大学大学院工学研究科教授。現在に至る。理学博士。IEEE（senior member），電子情報通信学会（シニア会員），ACM，情報処理学会，各会員。

藤　枝　直　輝（非会員） 2013 年東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程修了。博士（工学）。豊橋技術科学大学電気・電子情報工学系助教，愛知工業大学工学部電気学科講師を経て，2021 年より愛知工業大学工学部電気学科准教授。プロセッサアーキテクチャ，FPGA 応用，組込みシステム，ディジタルシステム教育の研究に従事。情報処理学会，電子情報通信学会，IEEE 各会員。