電気学会論文誌 C（電子・情報・システム部門誌）
IEEJ Transactions on Electronics, Information and Systems
Vol.139 No.9 pp.952–957 DOI: 10.1541/ieejeiss.139.952

論　文

LegUp と oLLVM による難読化制御論理回路の実装

非会員　山田翔太郎*　正　員　市川　周一**a)　非会員　藤枝　直輝**

Implementation of Obfuscated Control Logic Circuit with LegUp and oLLVM

Shotaro Yamada*, Non-member, Shuichi Ichikawa**a), Member, Naoki Fujieda**, Non-member

（2019年1月6日受付，2019年6月5日再受付）

It is an important issue to protect the intellectual property of software. The authors proposed to conceal some part of software by implementing it as logic circuit. Though the security is further improved by obfuscating the logic circuit, it requires much effort to develop the dedicated obfuscation tool. Matsuoka et al. proposed using the software obfuscation tool, Obfuscator-LLVM (oLLVM), with C-backend (CBE) and Xilinx Vivado HLS (high-level synthesis) to generate the obfuscated logic circuit. This study proposes adopting another HLS, LegUp, to obfuscate logic circuit. The feasibility of these two methods are examined with 12 applications of CHStone benchmark, each of which were evaluated with four obfuscation methods (total 48 cases). In our experiments, Matsuoka's method failed to generate the correct hardware in 5 cases out of 48 cases. Meanwhile, the proposed method successfully generated the correct hardware for all 48 cases. The average latency derived by the proposed method was 43% larger than that by Matsuoka's method. The logic scale derived by the proposed method was also 42% larger (LUT) and 112% larger (FF) than that by Matsuoka's method.

キーワード：知的財産，耐タンパ性，高位合成，LLVM, FPGA
Keywords: Intellectual property, Tamper resistance, High-level synthesis (HLS), LLVM, FPGA

1. まえがき

制御プログラムの知的財産流出は深刻な社会問題である。制御プログラムに組み込まれた技術やノウハウは多大な時間と労力の成果であり，盗用されれば企業活動に大きな障害となる。特にソフトウェアは複製や解析が容易であるため，剽窃や改変などの被害に遭いやすい。この様な脅威からソフトウェアを保護することは，重要な課題である。

ソフトウェア保護には，暗号化，鍵認証，難読化など多様な手法が存在する[1]。暗号化は転送や保存時に有効だが，利用時には解除（復号）するため万能ではない。鍵認証の例はパスワードやライセンスコードによる認証であるが，安全性は鍵の管理にかかっており，また認証をバイパスするような改変にも無力である。難読化は，プログラムを機能的に等価かつ構造や動作が複雑になるように変換し，解析や改竄を阻害する技術である[1]。著者ら[2]~[4]は，ソフトウェアの一部をハードウェア化することにより，処理内容を秘匿する研究を続けてきた。これらの手法にはそれぞれ長所短所があり，組合せることによって，より高いレベルの保護が実現できる。

以下，本研究では，ハードウェア化と難読化を併用した保護手法について検討する。本論文の構成は以下の通りである。2章で研究背景および関連研究を説明し，本研究の目的を述べる。次に3章で実装･評価の方法を説明し，4章で評価結果を報告する。最後に5章で結論と今後の課題を述べる。

なお本論文は，第23回知能メカトロニクスワークショップ (IMEC 2018) で発表した原稿[5]に大幅な加筆修正を施したものである。

2. 研究背景と関連研究

市川ら[2][3]は PLC (Programmable Logic Controller) の制

a) Correspondence to: Shuichi Ichikawa. E-mail: ichikawa@ieee.org
* 豊橋技術科学大学大学院 電気・電子情報工学課程
〒441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1
Electrical and Electronic Information Engineering Course, Toyohashi University of Technology
1-1, Hibari-gaoka, Tempaku, Toyohashi, Aichi 441-8580, Japan
** 豊橋技術科学大学 電気・電子情報工学系
〒441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1
Department of Electrical and Electronic Information Engineering, Toyohashi University of Technology
1-1, Hibari-gaoka, Tempaku, Toyohashi, Aichi 441-8580, Japan

御プログラムを論理回路化することにより，動作速度と秘匿性を高めることを提案した。しかし PLC 命令列から論理回路を生成するには専用プログラムが必要であり，実用化へのハードルは高かった。藤枝ら[4]は C 言語の制御プログラムから高位合成 (HLS) を用いて論理回路を生成し，得られたシステムの性能を報告した。高位合成の利用により，制御論理回路を生成する労力は大幅に削減される。

論理回路の解析は，ソフトウェアの解析より難しいにせよ，不可能ではない。より高度な保護を実現するには他の保護手法と組合せることが必要である。論理回路の難読化[6]は選択肢の一つであり，著者ら[7]も Opaque Predicates による論理回路難読化を実装した。しかし難読化手法ごとに専用ツールを自作することは，開発労力が大きく実用性に乏しい。既存のソフトウェア難読化ツールと高位合成技術を組み合わせることができれば，開発コストを大幅に削減することが可能になる。

汎用のソフトウェア難読化ツールは近年まで存在しなかったが，2015 年に Obfuscator LLVM[8] (oLLVM) が発表された。oLLVM はオープンソースのプロジェクトで，コンパイラ基盤 LLVM[9] 上に実現されているため，言語や実装プラットホームによらず難読化を実現することができる。さらに独自の難読化手法を実装する場合も，LLVM のフレームワークを利用することにより，開発労力を大きく削減できる。

松岡ら[10]は，oLLVM による難読化がソフトウェアのサイズや実行時間に及ぼす影響を評価した。難読化により，平均実行時間は 1.3〜10.5 倍，平均コードサイズは 1.6〜2.5 倍に増大した。さらに松岡らは，oLLVM により難読化した C コードから難読化論理回路を高位合成し，平均論理規模が難読化により 1.1〜1.7 倍に増大すると述べた。難読化論理回路の処理時間については，不正確な見積値を示すに留まっている。

松岡ら[10]の研究には，幾つかの問題点がある。第一に，松岡らの評価では難読化論理回路の実装（論理合成）を行っていない。高位合成が出力する見積値に基づいて議論しているが，正確な評価とは言えない。実装結果に基づく，より正確な評価が必要である。第二に，高位合成後の論理シミュレーションが行われていないため，得られた回路の正常動作が確認されていない。CAD の出力が正しく動作することは当たり前のようだが，各 CAD ツールは利用方法が決まっており，前提から外れた使い方をした場合に正しい出力が得られる保証はない。大規模ソフトウェアである以上，バグによって誤動作する可能性もある。これらが杞憂でないことは，〈4·1〉章で具体的に示す。第三の問題は，難読化論理回路の生成経路である。松岡らは oLLVM の出力を C 言語に変換してから高位合成しているが，難読化後に C 言語を経由することが問題である。ソフトウェア難読化においては oLLVM による難読化オーバヘッドは平均 1.3〜10.5 倍であるが，oLLVM の出力を一度 C 言語に変換するとオーバヘッドは平均 2.3〜12 倍に増加する[10]。ハードウェア難読化においても，難読化後に C 言語を経由することにより性能劣化が生じる可能性がある。この点について検討が必要である。

本研究の目的は，以上 3 つの問題について検討し，評価結果を示すことである。第一の問題については，高位合成後に論理合成を施し，正確な論理規模を測定する。第二の問題については，論理シミュレーションで難読化制御論理の動作を確認し，同時に実行時間（サイクル数）を測定して実行時間のオーバヘッドを正確に評価する。第三の問題については，oLLVM の出力から C 言語を経由せずに高位合成を行い，松岡らの手法と評価結果を比較する。

3. 実装評価手法

〈3·1〉 実装方法 本研究における処理手順を Fig. 1 にまとめる。実線が本研究の処理手順，破線は比較対象とする松岡ら[10]の処理手順である。以下の評価で使用する各ソフトウェアのバージョンも，合わせて Fig. 1 に示した。

処理手順の前半部分はコンパイラ基盤 LLVM[9] 上に実現されており，言語や実装プラットホームによらず難読化を実現することができる。LLVM では，ソースコードは言語フロントエンドで中間表現 (LLVM IR) に変換される。Fig. 1 において，Clang は C 言語のフロントエンドである。生成された IR は，“パス” と呼ばれるモジュールによって解析や最適化が行われる。このパスはユーザが任意に組み合わせることができ，独自のパスを実装することも可能である。oLLVM[8] は IR 形式のプログラムを難読化するパスで，難読化されたプログラムを IR 形式で出力する。

松岡ら[10]の処理手順（Fig. 1 の右側）では，oLLVM の出力（難読化された IR）を C バックエンド (CBE) で処理して C 言語記述に戻す。CBE には Julia Computing の C Backend[11] を採用している。Vivado HLS の入力は C 言語なので，高位合成に Vivado HLS を利用するためには CBE が必須である †。得られた難読化 C プログラムを Vivado

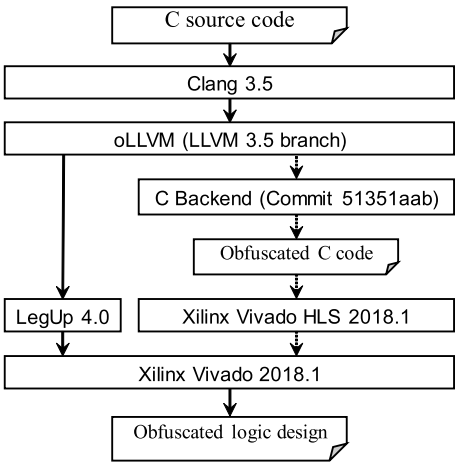


Fig. 1. Flowchart of hardware obfuscation.

† Vivado HLS は商用のため内部情報が非公開で，CBE なしで oLLVM と組み合わせる方法は知られていない。

Table 1. CHStone programs.[13]

Program	Design Description
dfadd	Double-precision floating-point addition
dfmul	Double-precision floating-point multiplication
dfdiv	Double-precision floating-point division
dfsin	Sine function for double-precision floating-point numbers
mips	Simplified MIPS processor
adpcm	Adaptive differential pulse code modulation decoder and encoder
gsm	Linear predictive coding analysis of global system for mobile communications
jpeg	JPEG image decompression
motion	Motion vector decoding of the MPEG-2
aes	Advanced encryption standard
blowfish	Data encryption standard
sha	Secure hash algorithm

HLS で高位合成し，難読化論理回路の Verilog 記述を得る。Vivado HLS の最適化オプションはデフォルト値とした。

本研究の処理手順（Fig. 1 左側）では，オープンソースの高位合成ツール LegUp[12] を使用する。LegUp は LLVM 上で実装されており，LLVM IR を入力として高位合成することができる。オープンソースなので処理の自由度が高く，他のツールと組合わせることも容易である。LegUp を採用することで，処理手順から CBE を排除し，難読化論理回路の論理規模と遅延を削減できる可能性がある。

LegUp の最新版は 6.4 であるが（2018 年 12 月現在），5.0 以降は商用ソフトウェアとなっており利用条件が厳しい。そこで本研究では，研究利用に適したオープンソース版 (LegUp 4.0) を評価に使用した。

松岡ら[10] は高位合成の出力（見積値）で論理規模と遅延時間を議論したが，本研究では高位合成の出力を Xilinx Vivado で論理合成し，より正確な評価を行う。論理シミュレーションを行うことにより，遅延時間をサイクル単位で正確に測定できる。LegUp で生成した論理回路だけでなく，CBE と Vivado HLS でも回路を生成して定量的比較を行う。Vivado の最適化オプションはデフォルト値とし，ターゲットデバイスは Xilinx Virtex UltraScale+ (xcvu9p-flgb2104) とした。

評価対象となる C 言語プログラムには CHStone ベンチマーク[13] (v1.11) を採用する。CHStone は高位合成用のベンチマークであり，Table 1 に示す 12 種のプログラムで構成されている。CHStone は YXI 社の HLS 製品 eXCite をターゲットとしているため，他の HLS 製品で利用する場合は適宜書き換えが必要である。松岡ら[10] は Vivado HLS 用に CHStone を修正したので，本研究でも松岡らが修正したソースコードを使用した。

〈3·2〉 難読化手法 現在の oLLVM には，偽の制御フロー，制御フロー平坦化，命令置換の 3 種類の難読化が実装されている。

偽の制御フロー (Bogus Control Flow) は Collberg らが提案した Opaque Predicates[14] を拡張した難読化である。Opaque Predicates は実行されない分岐を追加する事で，プログラムの複雑化を実現する。oLLVM の `-mllvm -bcf` オプションで，偽の制御フローを付加することができる。本論文でプログラム名の後ろに “_b” を付加した場合，偽の制御フローが付加されたプログラムを表すこととする。例えば adpcm というプログラムに偽の制御フローを施したものは，adpcm_b と表記する。

制御フロー平坦化[15] (Control Flow Flattening) は制御構造を分岐により平坦化し，アルゴリズムの特徴を読み取りにくくする手法である。本文中では，アプリケーション名の後ろに “_f” を付加することにより，制御フロー平坦化が施されたアプリケーションを表す。制御フロー平坦化は oLLVM の `-mllvm -fla` オプションで利用可能である。

命令置換 (Instructions Substitution) は，算術または論理演算子を機能的に等価な命令列に置き換えることである。本論文でアプリケーション名の後ろに “_s” を付加した場合，命令置換が適用されたアプリケーションを表すこととする。命令置換は oLLVM の `-mllvm -sub` オプションで利用可能である。

本研究では CHStone の各プログラムについて，難読化しない，偽の制御フロー，制御フロー平坦化，命令置換，の 4 通りを評価する。複数の難読化手法を適用することも概念上は可能であるが，試してみるとエラーが多く発生し，現実的評価は困難だった。oLLVM が複数手法の適用を想定していないか，バグによって動作しないものと考えられる。適用の回数と順番も考慮すると組合せは非常に多くなり，網羅的に検討することは難しい。そこで本研究では，難読化手法は 1 つだけを適用するものし，複数手法の適用については今後の課題とする。

4. 評価結果

〈4·1〉 変換結果の検証 まず難読化論理回路の生成と動作確認の結果について報告する。CHStone の 12 プログラムについて，それぞれ 4 つの難読化オプション，計 48 通りを調べた。結果は Table 2 にまとめた通りである。松岡らの処理手順は CBE+VivadoHLS，本研究の処理手順は LegUp と表示している。

CHStone の各プログラムはテストベクタを内蔵しており，実行結果が正しいかどうか自己判定している。表中，“–” で示された部分は，正常に回路が生成され，実行結果が正しかったことを示している。“Fail” は，回路はエラーなしに生成されたが，シミュレーションで誤った結果が得られたことを示す。

CBE+VivadoHLS については，色々な問題が発生した。blowfish_b は高位合成に失敗して難読化論理回路を生成することができなかった。直接的には，Vivado HLS が CBE の出力を処理できなかったことになるが，その原因が oLLVM, CBE, Vivado HLS のいずれにあるか不明である。松岡ら[10] は blowfish_b の高位合成に成功しているが，本研究とは各ソフトウェアのバージョンが異なるため結果に差が出たと思われる。motion_b，blowfish_f では，シミュレーションの結果が期待値と異なっていた。gsm_f と jpeg_f では，得られた回路の論理シミュレーションが終了しなかっ

Table 2. Result of benchmark tests

Program	LegUp	CBE+ VivadoHLS	Program	LegUp	CBE+ VivadoHLS
adpcm	–	–	dfsin	–	–
adpcm_b	–	–	dfsin_b	–	–
adpcm_f	–	–	dfsin_f	–	–
adpcm_s	–	–	dfsin_s	–	–
aes	–	–	gsm	–	–
aes_b	–	–	gsm_b	–	–
aes_f	–	–	gsm_f	–	Simulation stuck
aes_s	–	–	gsm_s	–	–
blowfish	–	–	jpeg	–	–
blowfish_b	–	HLS failed	jpeg_b	–	–
blowfish_f	–	Fail	jpeg_f	–	Simulation stuck
blowfish_s	–	–	jpeg_s	–	–
dfadd	–	–	mips	–	–
dfadd_b	–	–	mips_b	–	–
dfadd_f	–	–	mips_f	–	–
dfadd_s	–	–	mips_s	–	–
dfdiv	–	–	motion	–	–
dfdiv_b	–	–	motion_b	–	Fail
dfdiv_f	–	–	motion_f	–	–
dfdiv_s	–	–	motion_s	–	–
dfmul	–	–	sha	–	–
dfmul_b	–	–	sha_b	–	–
dfmul_f	–	–	sha_f	–	–
dfmul_s	–	–	sha_s	–	–

た。シミュレータで調べると，2 つの状態の間を往復しており処理が進まない状況であった。いずれも誤った回路が生成されたことを示唆するが，原因は不明である。

これらの異常については原因を究明してバグを修正することが望ましいが，本研究の目的は「良い処理手順を発見すること」である。各処理手順の問題点や得られる回路の質を把握することを優先し，各ソフトウェアのデバグは基本的に今後の課題とした。

LegUp については，48 通り全ての難読化回路が正常に生成され，正常に動作した。この点については，明らかに CBE+VivadoHLS の処理手順より優れている。ただし，LegUp については，幾つかの問題を修正して使用している。

- Xilinx 向けメモリ初期化ファイル (MIF) の生成に問題があったため修正した。この修正前は，adpcm を高位合成した際にシミュレーションで誤動作 (Fail) していた。
- LegUp が出力するデュアルポート RAM 記述を Vivado が LUT と FF で合成するという問題を修正した。LegUp の RAM 記述テンプレートを書き換えて，Vivado が BRAM を使用するようにした。
- LegUp が 2 ベキの除算でも除算器を生成するため，2 ベキを別扱いするよう修正した。

これらは LegUp の問題ではあるが，実際には Xilinx デバイスに固有の問題である。LegUp は Altera FPGA を開発基盤としているため，Xilinx デバイスのサポートに難が残る。一方の Vivado HLS は Xilinx 純正のツールであるため，Xilinx デバイスで良い結果を得やすい。この点を踏まえて，上記の修正を施すことは不公平に当たらないと考え，修正版の LegUp を評価に用いた。

他にも，制御フロー平坦化を行った際，LegUp が高位合成に失敗するという問題があった。これについて調査した結果，原因は oLLVM のバグであったため，バグを修正した oLLVM を評価に用いた。LegUp だけでなく CBE+VivadoHLS の評価でも同じ oLLVM（バグ修正済）を用いている。CBE+VivadoHLS は制御フロー平坦化で問題を抱えているが (Table 2)，それは上記のバグとは別の問題である。

このように各種の問題を修正できることは，オープンソースのツールを採用する利点である。商用 CAD ツールを利用する場合，内部情報が公開されておらず，またライセンス条項で解析が禁じられている場合もあるため，問題への対処が難しい。その意味でもオープンソースで完結する本研究の処理手順には一定の優位性が認められる。

〈4·2〉 論理規模と遅延時間 本章では，生成された難読化論理回路の論理規模と遅延時間について述べる。FPGA は，LUT (look-up table)，FF (flipflop)，BRAM (block RAM), DSP (乗算器) など，目的の異なる様々な構成要素からなる。従って論理規模を単一の指標で表すことは適切でない。以下の評価でも，構成要素の種類別に規模を示す。

Fig. 2 は，提案手法 (LegUp) で生成した回路の論理規模と遅延時間を，既存手法 (CBE+VivadoHLS) で生成した回路からの増分 (%) でグラフ化したものである。blowfish_b は既存手法で回路生成に失敗したため省かれている。blowfish_f と motion_b は，既存手法で難読化論理回路が正しく動作していないが，参考値として掲載している。gsm_f と jpeg_f は，既存手法で論理シミュレーションが終了せず，遅延時間が不明である。そこで論理規模だけを参考値として掲載し，遅延時間は省略している。

Fig. 2 で，論理規模と遅延時間は小さい方が優れているので，増分が大きいということは提案手法が既存手法より劣っていることを表す。遅延時間の増分は −38〜+160% (平均 +42.7%)，LUT の増分は −60〜+200% (平均 +41.6%)，FF の増分は −50〜+302% (平均 +112%) であった。BRAM と DSP については既存手法で未使用の場合が多く，比で表せないので Fig. 2 には載せていない。BRAM の使用量は，既存手法で 0〜51 個（平均 11.0 個），提案手法で 0〜36 個（平均 7.2 個）である。BRAM の平均値でみると提案手法の増分は −34.4%となり，BRAM を少なく LUT/FF を多く使用する傾向にある。同様に DSP の使用量は，既存手法で 0〜104 個（平均 20.0 個），提案手法で 0〜144 個（平均 31.7 個）である。DSP の平均値では提案手法の増分は +58.3%となる。〈4·1〉章でも述べた通り，LegUp は BRAM を十分に利用できず，また演算器を多く消費する傾向が伺える。

このような傾向は，難読化に起因するものではない。Fig. 2 で難読化されていない回路 (12 種) だけを調べてみても，遅延時間は +47.7%，LUT は +43.3%，FF は +108%となっている。即ち遅延時間と論理規模の増加は，高位合成ソフトウェアの性能差によるものと考えられる。オープンソー

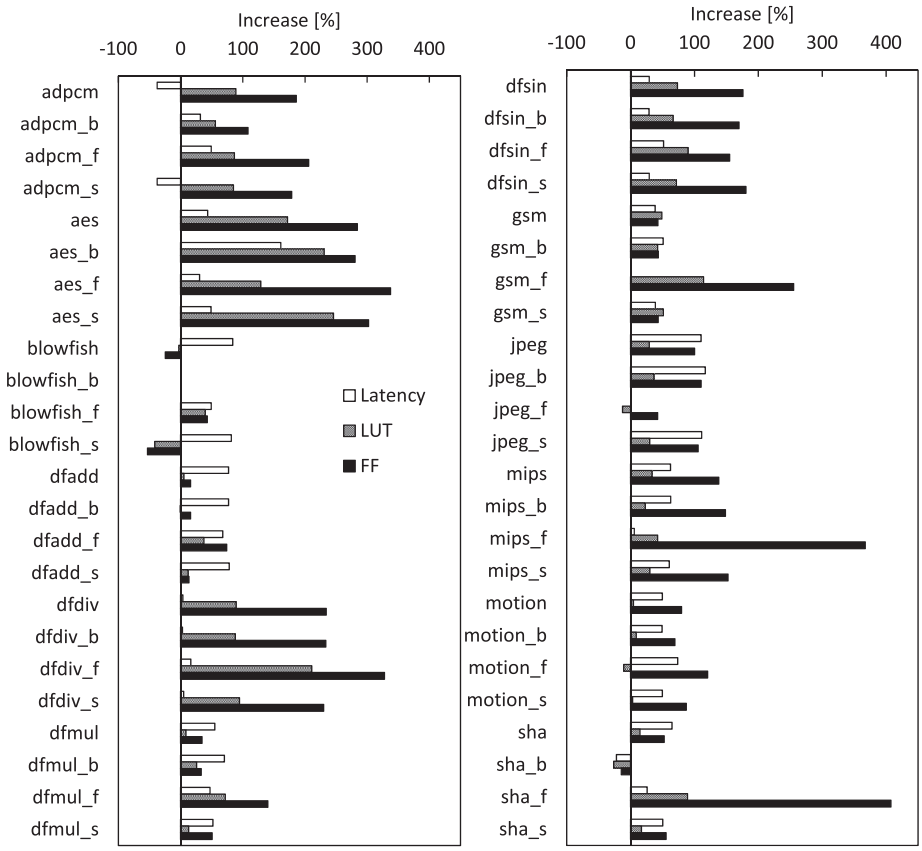
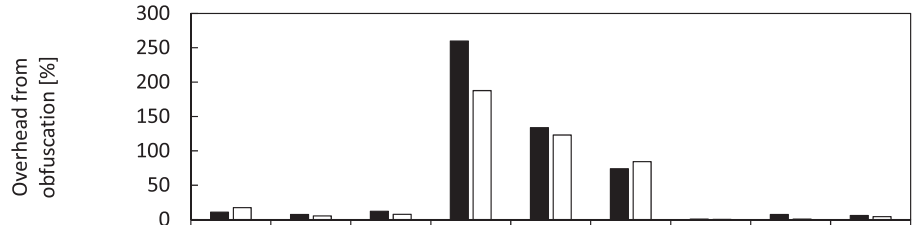


Fig. 2. Increase in latency, LUT, and FF



	Latency (b)	LUT (b)	FF (b)	Latency (f)	LUT (f)	FF (f)	Latency (s)	LUT (s)	FF (s)
■ CBE + Vivado HLS	10.6	7.3	12.0	259.8	133.8	73.8	0.6	7.2	5.9
□ LegUp	17.3	5.2	7.7	187.7	123.1	84.2	0.0	0.6	4.1

Fig. 3. Average overhead from obfuscation

スの LegUp は，現状，まだ商用の Vivado HLS に及ばないということである †。しかし高位合成技術の成熟とともに，実力の差は縮まると期待される。かつてフリーソフトの C コンパイラ (gcc 等) は商用 C コンパイラに及ばなかったが，現在では多くの局面で差はなくなっている。HLS でも同様の進歩が期待される。

Fig. 3 には，手法毎の難読化オーバヘッドについてまとめた。縦軸は増分であるが，難読化しない場合を基準として，難読化により遅延時間と論理規模がどの程度増大するか示したものである。制御フロー平坦化 (f) においては，既存手法と提案手法の差は少ないことがわかる。偽の制御フロー (b) と命令置換 (s) では，既存手法・提案手法ともにオーバヘッドがほとんどない。これは，松岡ら [10] が報告したように，HLS による解析と最適化によって難読化が除去されたものと考えられる。この点では，Vivado HLS も LegUp も同様の回路を生成していることが伺える。

† Nane ら [16] によるサーベイは，FPGA 用 HLS システムの機能や性能差について網羅的に報告している。

5. むすび

本研究では，LegUp と oLLVM を用いて CHStone ベンチマークから難読化論理回路を生成し，その遅延時間と論理規模を評価した。本研究により，汎用難読化ツール (oLLVM) と高位合成ツール (HLS) を用いて難読化論理回路が生成可能であることが実証された。これにより，難読化論理回路を容易かつ広範囲に利用することが可能になった。

既存研究 [10] では LegUp のかわりに CBE と Vivado HLS を用いていた。12 プログラム × 4 難読化手法（計 48 通り）

について評価した結果，既存手法 (CBE+VivadoHLS) では 5 つのケースで難読化論理回路が生成できない，あるいは正しく動作しないことが判明した。CBE 経由の処理は本質的に冗長なので，途中の変換で問題が生じやすいと考えられる。一方，提案手法 (LegUp) では 48 通り全てで正しく動作する難読化論理回路を生成することができた。直接 LLVM IR から難読化論理回路を生成することにより，既存手法における問題を回避することができた。

本研究では高位合成に LegUp を用いることにより，既存研究における (CBE の) オーバヘッドが削減されることを期待した。しかし遅延時間と論理規模については，提案手法の方が既存手法より大きくなることが確認された。これは使用した高位合成ソフトウェアの能力差によるものである。

既存手法の問題は，性能低下より，多くの場合に正しい回路が生成できない点であることが分かった。また，既存手法・提案手法のいずれにおいても，簡単な難読化が HLS により除去されることが確認された。これを解決するには，難読化論理回路の生成時に，HLS の最適化をバイパスする等の工夫が必要である。この点については今後の課題とする。

謝　辞

本研究の一部は JSPS 科研費 16K00072 の支援による。

文　献

（1） A. Monden and C. Thomborson: “Recent Software Protection Techniques - Software-only Tamper Prevention -”, IPSJ magazine, Vol.46, No.4, pp.431–437 (2005) (in Japanese)
門田暁人・Clark Thomborson：「ソフトウェアプロテクションの技術動向 (前編) - ソフトウェア単体での耐タンパー化技術」，情報処理，Vol.46, No.4, pp.431-437 (2005)

（2） S. Ichikawa et al.: “Converting PLC instruction sequence into logic circuit: A preliminary study”, Proc. 2006 IEEE Intl. Symp. Industrial Electronics (ISIE ’06), pp.2930-2935 (2006)

（3） S. Ichikawa et al.: “An FPGA implementation of hard-wired sequence control system based on PLC software”, IEEJ Trans. Electrical and Electronic Engineering, Vol.6, No.4, pp.367-375 (2011)

（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） S. Yamada, S. Ichikawa, and N. Fujieda: “An Experimental Implementation of Obfuscated Control Logic Circuit with LegUp”, 23rd Intelligent Mechatronics Workshop (IMEC 2018), 2B4-2 (2018) (in Japanese)
山田翔太郎・市川周一・藤枝直輝：「難読化制御論理回路の LegUp による試作」，第 23 回知能メカトロニクスワークショップ (IMEC 2018), 2B4-2 (2018)

（6） D. Forte, S. Bhunia, M.M. Tehranipoor (Eds.): Hardware Protection through Obfuscation, Springer International Publishing (2017)

（7） Y. Ishigaki, N. Fujieda, Y. Matsuoka, K. Uyama, and S. Ichikawa: “An Obfuscated Hardwired Sequence Control System Generated by High Level Synthesis”, Proc. Fifth International Symposium on Computing and Networking (CANDAR 2017) (2017)

（8） P. Junod, J. Rinaldini, J. Wehrli, and J. Michielin: “Obfuscator-LLVM-Software Protection for the Masses”, Proc. International Workshop on Software Protection (SPRO 2015), pp.3-9 (2015)

（9） C. Lattner and V. Adve: “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation, Proc. 2004 International Symposium on Code Generation and Optimization (CGO 2004), pp.75–86 (2004)

（10） Y. Matsuoka, N. Fujieda, and S. Ichikawa: “Evaluation of hardware obfuscation techniques using obfuscation tool oLLVM”, *IEEJ Trans. IA*, Vol.139, No.2 (to appear) (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 (to appear) (2019)

（11） Julia Computing: “llvm-cbe-fork (LLVM 3.5), github.com” https://github.com/JuliaComputing/llvm-cbe-fork (accessed 2017-07-25)

（12） A. Canis et al.: “LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems”, ACM Trans. Embedded Computing Systems, Vol.13, No.2, Article 24 (2013)

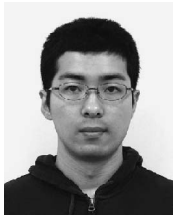
（13） Y. Hara, H. Tomiyama, S. Honda, and H. Takada: “Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis”, IPSJ Journal, Vol.17, pp.242-254 (2009)

（14） C. Collberg, C. Thomborson, and D. Low: “A Taxonomy of Obfuscating Transformations”, Department of Computer Science, The University of Auckland, Technical Report No.148 (1997)

（15） T. László and Á. Kiss: “Obfuscating C++ Programs via Control Flow Flattening”, Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Computatorica, Vol.30, pp.3-19, Hungary (2009)

（16） R. Nane et al.: “A Survey and Evaluation of FPGA High-Level Synthesis Tools”, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol.35, No.10, pp.1591-1604 (2016)

山　田　翔太郎　（非会員）　2019 年 3 月，豊橋技術科学大学電気・電子情報工学課程卒業。同年 4 月より同大学大学院工学研究科電気・電子情報工学専攻博士前期課程在学。電子情報通信学会会員。



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 創造科学推進事業 (ERATO) 後藤磁束量子情報プロジェクト研究員。1991 年三菱電機 (株) LSI 研究所，システム LSI 開発研究所勤務。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。並列計算機，並列処理，および専用計算システムアーキテクチャの研究に従事。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）。同年より豊橋技術科学大学電気・電子情報工学系助教。2019 年より愛知工業大学工学部電気学科講師。プロセッサアーキテクチャ，FPGA 応用，組み込みシステム，セキュアプロセッサの研究に従事。情報処理学会，電子情報通信学会，IEEE 各会員。