LogicLocking による FPGA 回路の階層的な保護を目的とした回路設計情報難読化の提案

北川　裕基[1]　中沢　実[1,a)]　河並　崇[1]

概要：本研究では、データセンタなどにおいてアクセラレーション用途としてインフラストラクチャ機能のハードウェアオフローディングが注目されてきている中で特に、FPGA に焦点を当て Bitstream のリバースエンジニアリングなどから回路の設計情報を保護するための階層的なセキュリティ保護を目的とする．難読化手法である LogicLocking を実装しサーバサイドにおける FPGA 活用におけるセキュリティを検討する．

キーワード：FPGA アクセラレーション, LogicLocking, セキュリティ, 難読化, ランダム LogicLocking

1. はじめに

近年，ネットワークの分野では，これまで Application Specific Integrated Circuit(ASIC) で実装され固定化されていたネットワークデータプレーン機能を仮想化のためにソフトウェアとして実装することで, 汎用的なサーバの上で動作可能なものが登場し, Software Defined Network(SDN) や Network Function Virtualization(NFV) が広まっていった．さらにデータセンタ内の East-West トラフィックが増大したことにより [3]，高性能化や低電力化が望まれ，プログラム可能なデータプレーンスイッチや SmartNIC が登場し，サーバのネットワーク機能のオフロードへの使用として大きな期待となる活用法となっている [13]．ハードウェアレベルでのプログラマビリティが向上した背景として主に Field Programmable Gate Array(FPGA) の発展が大きい．また，プログラム可能なデータプレーン専用の言語として P4 が成熟してきた [5]．開発面においても，AMD(旧 Xilinx) や Intel(旧 Altera) の SmartNIC または FPGA アクセラレーションボードには P4 を用いてネットワークデータプレーン回路を作成できるツールが備わっており，ソフトウェア技術者が開発しやすい環境が用意されている．実際に P4 で記述する部分だけをクライアント側が考え，最終的にボードに対して P4 から生成した回路とその他のハードウェアに関する実装部分の開発や組み合わせて合成する部分，アクセラレーターにプログラムする部分をベンダーとクライアントの仲介役として受け持つようなサービスを提供している取り組みが存在しており，開発支援という部分でも整えられてきている．SmartNIC 全体の市場は今後急成長していくと予測される [14].

本研究で想定している脅威モデルは，FPGA を対象として攻撃者によって Bitstream 暗号化が破られた，またはハードウェアに Bitstream 暗号化機能が存在していなかった場合において，元の回路をリバースエンジニアリングして得られるゲートレベルのネットリストにアクセスできることを前提としている．そこで階層的な防御アプローチとしての LogicLocking[2] を既存設計ソリューションと組み合わせたセキュリティ設計を行うことを考える．そしてネットワーク回路へ応用し，オープンには公開したくないネットワークデータプレーンの保護についても検討する．FPGA へのコンフィグレーションの際，セキュリティ保護として，Bitstream 暗号化のみでは，リバースエンジニアリングに対する防御策として直接的な対策とはならず，実際に Xilinx 7 Series/Virtex-6 FPGA に対する Bitstream 暗号化の脆弱性が USENIX Security ’20 にて Maik Ender らによって発見されている [4]．そのため，何らかの攻撃方法により Bitstream 暗号化が破られゲートレベルのネットリストがあらわになった場合でも難読化された回路であれば，攻撃者が回路の構成について情報を得ることは難しくなることからもこのアプローチは有効であると考える．また，Bitstream 暗号化がサポートされていない安価なデバイスへのセキュリティ保護としての提案としても役に立つと考えている．

さらに，サーバサイドシステムにおいて今後 FPGA を用いたアクセラレーションボードの活用が増加した際には，

[1] 金沢工業大学石川県野々市市扇が丘 7-1
[a)] nakazawa@infor.kanazawa-it.ac.jp

ネットワークを介して Bitstream を FPGA にプログラムする環境において対象のサーバに攻撃者が侵入が可能であり権限を取得している場合，FPGA に対してもアクセス可能であるため，ハードウェアトロイを埋め込むといったような可能性も想定されると考えている．

本論文では，初期に考案された XOR，XNOR をランダムに元の回路に挿入する方式のランダム LogicLocking の実装と検証を紹介する．

本論文の流れは，2 章で関連研究を挙げ，3 章にて提案手法と簡単な実装について述べる．4 章にてサーバサイドにて FPGA 活用を想定した LogicLocking を行う場合を検討する．5 章にてまとめと今後に向けて述べる．

2. 関連研究

2.1 LogicLocking または回路難読化

まずはじめは，ハードウェアセキュリティにおける難読化についての説明と LogicLocking について整理する．本研究において FPGA に焦点を当てたハードウェアセキュリティとして LogicLocking という分野があり，難読化技術の一つである．主に，IP(知的財産) の盗難，過剰生産，ハードウェアトロイ，リバースエンジニアリングに対する防御策として集積回路 (IC) サプライチェーンのセキュリティのために議論されている [2].

LogicLocking は，元の設計に対してハードウェアに搭載されている安全な改ざん防止メモリに保存された秘密鍵でロックすることにより，保護対象である回路のデザインを難読な回路に変更する．例として図 1 のような 1 bit の半加算器に対して，Logisim を用いて XOR を使用したゲートレベルで LogicLocking による難読化を行った回路に正しい Key 入力を行った際の状態として図 2 と正しくない Key 入力を行った際の状態として図 3 を示している．

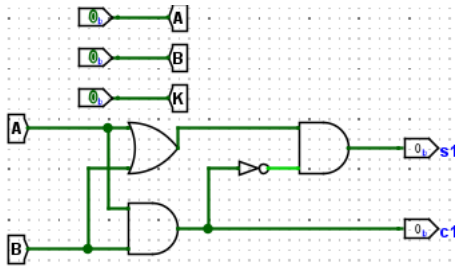

図 1 LogicLocking を行う前の 1 bit 半加算器回路

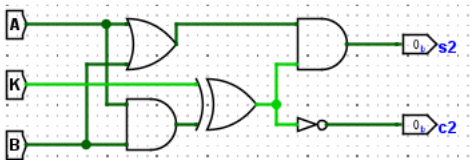

図 2 XOR による LogicLocking を行い正しいキーが設定されている例

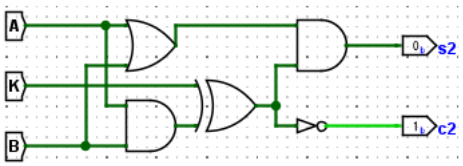

図 3 XOR による LogicLocking を行い誤ったキーが設定されている例

LogicLocking に関する進化の概観について様々な LogicLocking 方式の説明が IC，ASIC に限らず，FPGA，GPU といったアクセラレーションチップに対するロックなど体系化されまとめられている [2].

XOR/XNOR ゲートやルックアップテーブル (LUT) を用いた LogicLocking が主流であったころ，証明可能な検証ロジックはなくセキュリティの有効性に関して疑問視された．また，LogicLocking に用いる秘密鍵を総当たり的な特定によって回路の難読化を解除するために The Boolean Satisfiability(SAT) が使用された．現在研究されている多くの LogicLocking へのセキュリティ検証には，SAT や Satisfiability Modulo Theories(SMT) を用いた攻撃方法によるベンチマークが行われている．[2] では，LogicLocking について pre-SAT LogicLocking と post-SAT LogicLocking と大きく分類されており図 2 で示した LogicLocking は pre-SAT に割り当てられるため SAT 攻撃が通ると考えられる．一方 post-SAT として割り当てられる LogicLocking は SAT 攻撃に対する耐性が考えられたものとなっている．

先行研究 [6] では，FPGA 設計へのセキュリティフレームワークについて考えられており Register Transfer Language(RTL) から FPGA へのプログラムまでの間に LogicLocking を行う自動的な設計フローとネットワークに接続された再構成可能な IoT デバイスに対してロックされた回路を解除するためのセキュアなブート方法に取り組んでいる．

3. 提案手法

3.1 FPGA への難読化フローと実現方法について

本章では，LogicLocking を行うための手順と実装について説明する．使用する回路として，全加算回路を参考として使用している．

3.1.1 yosys

論理合成ソフトとして OSS(オープンソースソフトウェア) である yosys を使用して vivado などで作成した Verilog を論理合成し、下位のゲートレベルのネットリストを使用するために使用している．論理合成後のネットリストの出力には Verilog 形式のほかにも JSON 形式で出力可能である．

3.1.2 neo4j

組み合わせ論理回路の各ゲート構成をネットリストを元にグラフを作成するためにグラフデータベースである

neo4j を使用する．グラフデータベースでは，グラフを構成する要素となるノードの情報を定義しノードの作成と作成されたノード間の個々のリレーションを作成することで，多様なグラフ構造での情報の保存が可能である．これを用いてネットリストから組み合わせ回路部分のノード情報として論理ゲートのタイプと接続されているインプットまたはアウトプットといった情報をグラフとして反映する．グラフによって組み合わせ論理回路を構成する接続関係を表現し，その情報を元に難読化として LogicLocking を行う際の回路の編集に使用する．例として図 4 に示す．この図では，グラフデータベースを用いた際に neo4j の Web ダッシュボードから確認可能な図から見やすいように整形したものである．

オレンジのノードが入力または，出力ノードとなっており右端に並んでいるノード群が入力，左端に並んでいるノード群が出力となっている．紫のノードは，論理ゲートとなっており，OR や AND，XNOR，XOR，NOT などの一般的な組み合わせ回路を表している．水色のノードは，接続を表すノードであり Verilog では値を保持しない一時的な変数として wire として定義されるノードを表している．赤色の各ノードはインプットとなるノードが子，アウトプットとなるノードが親として回路情報を構成する．これらのノードやノード関係を難読化の際に変更することで回路を編集する．

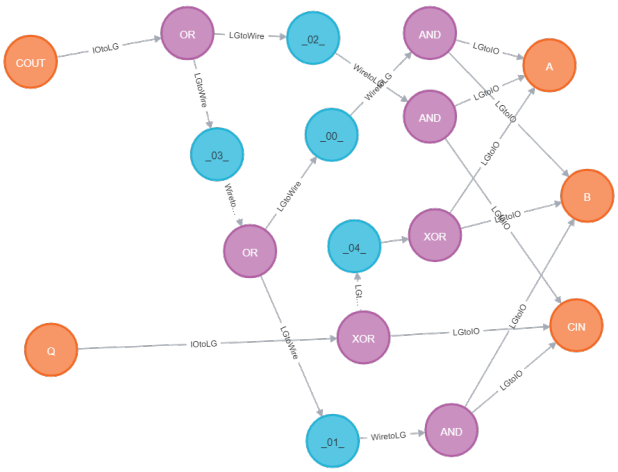

図 4　グラフデータベースでの回路構成の図

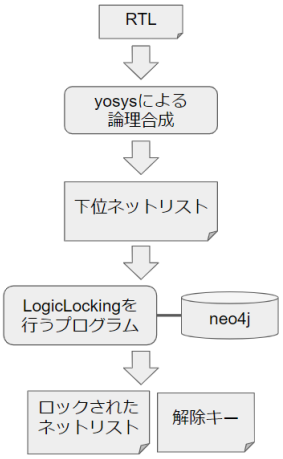

図 5　実装の概要図

ラミングしやすいまたは可読性が高い形としての記述がされているものを論理合成によって，FPGA に実装することができる形式に変換されると組み合わせ回路は 1 対 1 の接続と論理回路に変換されネットリストと呼ばれる．yosys では，論理合成後のネットリストを Verilog のほかに JSON などでも出力することができるため扱いやすく，難読化の際でも活用しやすいと考えている．今回の実装では，Verilog 出力を行い Pyverilog[8] という Verilog HDL 用のオープンソースのハードウェア設計処理ツールキットにて Verilog の構文解析を行い構文解析結果のテキストファイルを LogicLocking 用プログラムにて読み込み回路構成を取得している．今後はより実装のしやすさとネットリストのスキーマが比較的理解しやすいことを考慮し，yosys から直接ネットリスト形式の JSON を用いた回路情報の取得方法に置き換えるつもりである．

```
module fulladd( A, B, CIN, Q, COUT);
input A,B,CIN;
output Q, COUT;

assign Q = A ^ B ^ CIN;
assign COUT = (A & B) | (B & CIN) | (CIN & A);

endmodule
```

図 6　元となる全加算器の Verilog 記述

3.2　実装について

3.2.1　LogicLocking のための難読化フローについて

始めに全加算器を例として，実際に LogicLocking を回路に適用し難読化を行った際のフローを図 5 に示す．

回路をプログラムする際に作成する RTL について Verilog や VHDL または高位合成を用いて P4 や高級言語から生成された Verilog や VHDL でも対象としている．実際の元となる全加算器の Verilog を図 6 に示す．人間がプログ

3.2.2　難読化方法とランダム挿入ロックについて

以下に，LogicLocking による難読化を行うために作成したプログラムの流れを説明する．

(1) 回路の難読化方法として，事前に指定したキーの数と難読化スキーマを選択し LogicLocking を行うようにしている．方法として，XOR または XNOR を挿入する LogicLocking を行う．

(2) ランダムにサンプリングした論理ゲートのノードと接

続されている 1 つの出力接続関係を取得し，XOR または XNOR を間に挿入する．挿入した論理ゲートには元の接続とキーとなる入力ポートの作成と接続を行う．

(3) LogicLocking によりロックされた回路構成を作成した後は，キーと全体の回路をグラフからロック後のネットリストとして，Verilog を生成する．

(4) グラフ構造から回路構成を参照し，自動的にロックされたネットリストを生成するために，Verilog 構文への変換と Verilog ファイルの生成を行う．

挿入できるゲートと新しく作成できるキーの数は，ランダム XOR，XNOR の場合ロックを行う部分の回路全体のノード数以下の鍵空間となる．

参考として，ロックを施した後に生成したネットリストである Verilog の組み合わせ回路部分を図 7 に示す．また，このロックされた回路の正しいキーとして，key0 は 1，key1 は 0 が正しいキーとなっている．

```
assign COUT = _02_ | _03_;
assign key_gate_0 = CIN ^ _04_;
assign _04_ = B ^ A;
assign _00_ = B & A;
assign key_gate_1 = CIN & B;
assign _03_ = _01_ | _00_;
assign _02_ = A & CIN;
assign Q = ~(key_gate_0 ^ key_0);
assign _01_ = key_gate_1 ^ key_1;
```

図 7 ロック部分が挿入されている部分の Verilog 記述

3.2.3 生成した難読化回路の検証について

はじめに元の回路とロック後の回路が正しくキーを入力した場合等価な回路であるかという検証を行う．回路シミュレーターを用いた回路検証を行った際の回路を示す．元となった全加算器の回路図を図 8 に示し，ランダム XOR，XNOR 方式の LogicLocking 行った回路図を図 9 に示した．検証の結果，キーが正しい場合等価な回路となっており，キーが違う場合異なる出力となることが確認できる．

次に，ロックされたネットリストである生成を行った Verilog を Xilinx の開発ツールである Vivado にて読み込み検証を行った．Vivado では，スキーマを確認することができ正しく Verilog が読み込まれ意図した組み合わせ回路を生成することができているか確認することができた．図 10 に試した際のスキーマを示す．

次に，Vivado 側のシミュレートも行いツール上での動作を確認する．LogicLocking によって難読化された回路に対して解除を行うキーが正しい場合とキーが一部誤っている場合において出力結果がどのように変化するかを確かめる．

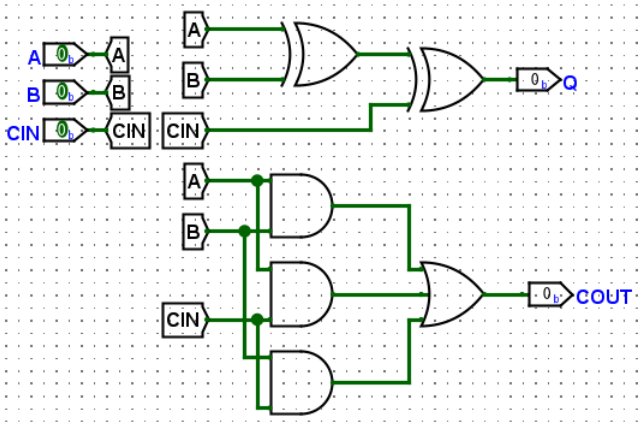

図 8 元の全加算器回路を回路シミュレーターにて再現した図

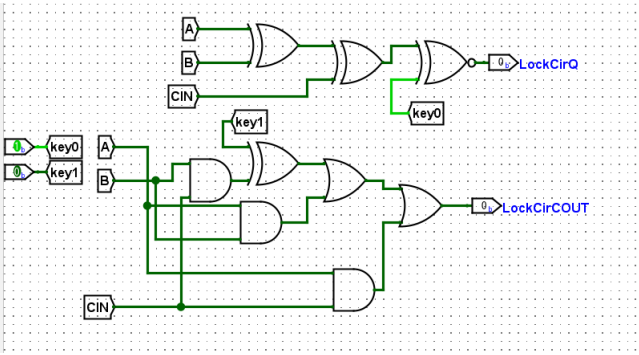

図 9 ロック後の回路を回路シミュレーターにて再現した回路

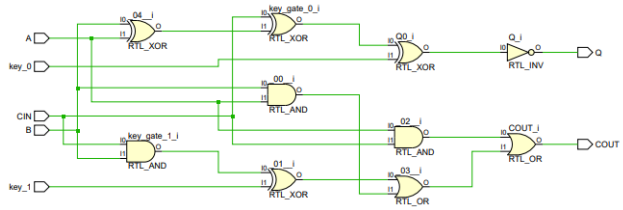

図 10 ロックされたネットリストを vivado からみた構造図

ベンチマーク用の Verilog を作成し全パターンの結果を波形シミュレーターに出力した結果として，正しいキーの場合を図 11，誤ったキーの場合を図 12 に示した．出力ポートである COUT と Q の出力シミュレーションの結果について一部変化していることがわかる．出力ポート Q では入力から出力に至るまでの経路上に存在するキーは正しいため元の出力と一致するが，出力ポート COUT では経路上に存在するキーに誤りがあり元の出力とは異なる場合がある．キーの規模や入力と出力のビット幅が大きければそれだけ LogicLocking を行った際と元の出力は大きく異なり難読性が増すことになる．しかしながら，回路規模が小さく難読化解除キーの長さも短いため，より大きな回路で実装を進める．

3.2.4 セキュリティ検証

LogicLocking に対する難読化の際に生成される解除キー

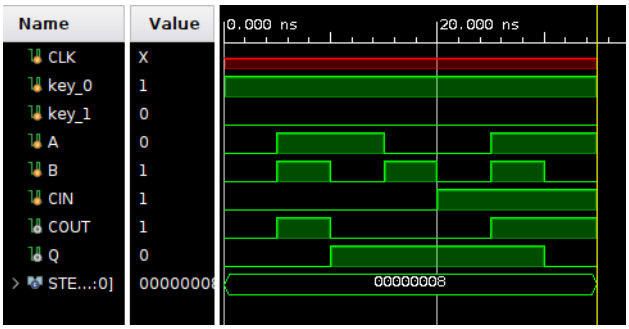

図 11 正しいキーの場合に vivado にてシミュレートした際の信号

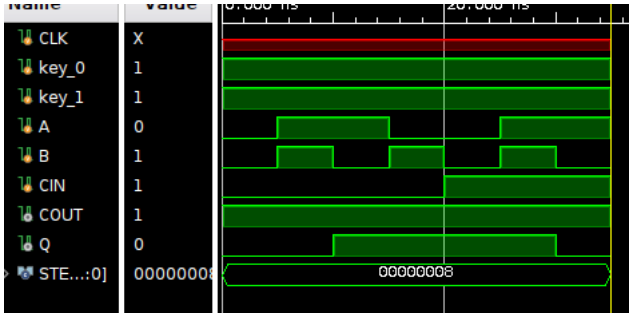

図 12 誤ったキーの場合に vivado にてシミュレートした際の信号

を正しく見つけ出すには，いくつかの過程において再構築しようとすることが挙げられているがその中で主に共通している 3 つを例として挙げる．

(1) 攻撃者は，ゲートレベルのネットリストの形でロックされた設計にアクセス可能である．
(2) キーとなる入力部分がわかっている．
(3) ロックされたネットリストの LogicLocking の方式が既知である．

このことから，攻撃者は手に入れた Bitstream からリバースエンジニアリングをある程度成功しており，その先の回路機能の詳細や悪意のある編集が困難となるようにしなければならない．

Bitstream の現実的なリバースエンジニアリングに関して，Tao Zhang らの論文から詳しくフローを見ることができる [10]．しかしながらソフトウェアのリバースエンジニアリングと異なり，未だ汎用的に使用できる Bitstream のリバースエンジニアリングツールは存在しておらず，かつ自力では難しいため本研究では取り扱うことはしないことにしている．将来的に需要の拡大と研究により解析が進むことは十分に考えられる．

今回，全加算器回路への LogicLocking としてランダムに XOR または XNOR を挿入する方式の実装行い動作の検証を行った．挿入ロックの方式は，XOR や XNOR の他に MUX や LUT を用いた方式が提案されているが，いずれもセキュリティ強度としては危ういということがベンチマーク回路での SAT を用いた攻撃手法によって [7] の研究などによって明らかになっている．そのため，SAT 耐性のあるロッキング手法が登場している．

SAT 耐性とは，SAT ソルバによってロックされた回路の難読化解除キーを見つけるまでにかかる時間が非常に多い場合において耐性があるとしている．また，主に解析実行にかかる時間は攻撃者の手に回路情報がわたり解析に使用される時間となることが考えられる．

今回実装として挙げた LogicLocking ではセキュリティの強度としては，心許ないため新しいロッキングスキームの実装にも現在取り組んでいる．

3.2.5 ロックした回路のためのキーのセキュアブートについて

LogicLocking によって難読化されロックされた回路を解除するためのキーを設定する場合，キーを配置するためのセキュアな方法が必要となる．通常，耐タンパー性メモリ内にキーを設定することによってキー自体の安全性を保ち難読化された回路を安全に起動を行う方法を取られているが，FPGA を用いたサーバサイドシステムでは，アクセラレーションボードとして活用されるためサーバ側からキーが提供されることや，またそのサーバに対して遠隔からキーを配布する際にネットワークを使用した場合においてそれらの安全なキーのやり取りも考慮しなければならない．

4. データプレーン保護のための LogicLocking の検討

FPGA 活用を考慮した際の，適用例としてネットワーク高速化回路を構成するデータプレーン回路保護のための LogicLocking についての考えを述べる．LogicLocking を用いた回路の難読化に関しては，ASIC などの物理的なチップ製造におけるサプライチェーンに対するセキュリティとして提案され始めたものであるが，今後サーバサイドや IoT デバイスにおける FPGA の活用機会が増し，Bitstream に対するリバースエンジニアリングと回路情報の解析による IP の侵害やハードウェアトロイの脅威もあると考える．

サーバサイドのネットワーク高速化として，NIC(Network Interface Card) の機能を FPGA にハードウェアオフロードし機能の追加など進められており設計や FPGA プログラムの段階でのセキュリティ脅威として対策が必要になってくると考えている．AMD では，Alveo シリーズにて使用できる FPGA ベースの NIC プラットフォームである OpenNIC[9] を提供しており，NIC シェル，Linux カーネルドライバー，DPDK ドライバーといった複数のコンポーネントにて構成されている．OpenNIC には，NIC 機能のほかにユーザーロジック部分が用意されているため自身で作成したネットワークデータプレーン回路と NIC を統合することができるようになっている．また，P4[5] を用いた高位合成を行いネットワークデータプレーン回路の作成

をハードウェア設計知識があまりなくても実装することができるツールセットである Vitis Networking P4[1] も提供されている．

しかしながら今回の LogicLocking による難読化を施そうとした場合，P4 からコンパイルされ生成された SystemVerilog 形式の RTL には，Xilinx の IP が含まれるため LogicLocking のようなことを外部のユーザーが変更を加えることはできない．そのため P4 から生成される回路と共に使用するユーザー独自のカスタムロジックなどに対してのみ適用することを前提に考えている．また，複数のモジュールからなる回路構成において，LogicLocking によって難読化し隠したいモジュール部分とそうでないモジュール部分のネットリストを統合させる，部分的な保護についても考慮することも重要であると考えている．

5. まとめと今後の展開

本論文では，今後サーバサイドにてデータセンタ向け FPGA アクセラレーションボードを用いる機会が増加していくであろう中で，回路情報の保護を行いリバースエンジニアリングやハードウェアトロイといったセキュリティ脅威に対する対抗策として，物理的なチップ製造において検討され続けている難読化手法の一つである LogicLocking に注目した．

回路の設計の際に C 言語や C++を用いて実装を行う高位合成 (HLS) を用いることがあり，高位の層で難読化を行う研究 [11][12] も存在するが，本研究では最適化の影響が比較的少ないであろうと考えられる点からゲートレベルでの難読化について取り組んだ．

ランダム挿入ロックとしてのゲートレベルでのランダム XOR，XNOR 挿入ロジックロックを行うためのプログラムの作成と Xilinx デバイスを視野に入れた設計ツールでのシミュレーション結果を確認しロックを行いたい回路に対して正しく LogicLocking が行われる実装ができるか検討を行った．結果として，非常に小さな回路として全加算器を元に LogicLocking を行い，意図したとおりに難読化された実装が可能であることを確認した．

今後，より複雑な回路やビット幅が大きい回路でも LogicLocking を行い検証を進める，またセキュリティ強度を考えた際に今回の実装を行ったランダム XOR，XNOR 方式の挿入ロックではなく SAT 耐性のあるロック方式や新しく提案されたロック方式をいくつか実装し検証することを現在取り組んでいる．

また，大きな回路を作成し FPGA にプログラムを行う場合では複数のモジュールが FPGA の中に構成される．その際には，自らが作成したモジュールとその他のモジュールが組み合わさったものとなることも考えられる．可能性の一つとしては，オープンソースによってコード全体や回路構成がすべて閲覧可能であるモジュールと企業などが提供している IP モジュールを使用する場合，企業などが提供するものはそもそも回路情報を確認することができない場合が存在するため，すべての論理ゲートに対して LogicLocking を施すことは現実的ではないと考えられる．

そのため，LogicLocking によってロックしたいモジュールとそうでないロックしなくてもよいモジュールが混在する設計となり，そのような回路において部分的に LogicLocking による難読化を可能にしていく．これにより LogicLocking をベンチマーク回路だけでなく実際の回路に対してロックを試すことができるように実装面を強化し，より実践的な評価が行えるようにしていく必要があると考えており本研究内容が貢献できると考えている．

さらには，複数 FPGA アクセラレーションボードが搭載されリコンフィギュレーションによる再プログラムが行われる可能性のあるデータセンタなどの環境でのセキュアなブート方式についても検討していきたい．

参考文献

[1] AMD. Vitis networking p4, 2024. https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html.

[2] Abhishek Chakraborty, Nithyashankari Gummidipoondi Jayasankaran, Yuntao Liu, Jeyavijayan Rajendran, Ozgur Sinanoglu, Ankur Srivastava, Yang Xie, Muhammad Yasin, and Michael Zuzak. Keynote: A disquisition on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):1952–1972, 2020.

[3] Kip Compton. Cisco's global cloud index study: Acceleration of the multicloud era, 2024. https://blogs.cisco.com/news/acceleration-of-multicloud-era.

[4] Maik Ender, Amir Moradi, and Christof Paar. The unpatchable silicon: A full break of the bitstream encryption of xilinx 7-series FPGAs. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1803–1819. USENIX Association, August 2020.

[5] Open Networking Foundation. P4language, 2024. https://p4.org/.

[6] Sam Reji Joseph. *LOGIC LOCKING FRAMEWORK AND ITS APPLICATION FOR SECURE BOOT IN FPGAs*. PhD thesis, The University of North Carolina at Charlotte, 2020.

[7] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[8] Shinya Takamaeda-Yamazaki. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *Applied Reconfigurable Computing*, volume 9040 of *Lecture Notes in Computer Science*, pages 451–460. Springer International Publishing, Apr 2015.

[9] Xilinx. Github repository: Amd opennic project, 2024. https://github.com/Xilinx/open-nic.

[10] Tao Zhang, Jian Wang, Shize Guo, and Zhe Chen. A comprehensive fpga reverse engineering tool-chain: From bitstream to rtl code. *IEEE Access*, 7:38379–38389, 2019.

[11] 翔太郎 山田, 周一 市川, and 直輝 藤枝. Legup と ollvm による難読化制御論理回路の実装. **電気学会論文誌 C（電子・情報・システム部門誌）**, 139(9):952–957, 09 2019.

[12] 小倉幹也 and 市川周一. Obfuscator-llvm と bambu を用いたハードウェア難読化手法. *VLD2023-54,ICD2023-62,DC2023-61,RECONF2023-57*, pages 125–130, 2023.

[13] 真壁 徹 and 宇多 仁. In-network computing から見る データセンタネットワークの研究動向と課題. In **信学技報**, *vol. 120, no. 125, IN2020-11, pp. 13-18, 2020* **年** *8* **月**., 2020.

[14] 日本経済新聞. ザイリンクス、クラウドサービスプロバイダー・通信会社向け smartnic プラットフォームを発表, 2020. https://www.nikkei.com/article/DGXLRSP530209_U0A300C2000000/.