Vol.2019-ARC-238 No.8
2019/11/15

高位合成用DSLコンパイラを用いたSLAMアプリケーションのハードウェアアクセラレーション

原 凌司[1,a)]　谷本 輝夫[1,b)]　井上 優良[2]　大澤 隆志[3]　丸岡 晃[3]　飯塚 拓郎[4]　追川 修一[5]
井上 弘士[1,c)]

概要：自己位置を推定するための手法として SLAM (Simultaneous Localization and Mapping) が用いられる．SLAM は周囲の環境の地図の作成／更新や地図内の自分の位置を推定するために，取得画像から特徴点を検出する機能やそれらを追跡する機能を持つ．これらの機能は SLAM での処理の大部分を占める．画像処理など定型処理の実行効率を高める手段として FPGA (Field-Programmable Gate Array) によるハードウェアアクセラレーションが注目されている．しかしながら，回路規模の問題から SLAM アプリケーション全体を FPGA 実装することは難しい．そこで，本研究では SLAM の実装の 1 つである ATAM を対象に機能の一部を FPGA 実装した．ATAM 主要機能のうち 4 機能のハードウェア化において，資源制約下でフレームレートを最大化するため，Halide DSL からハードウェアを生成可能なコンパイラを用いて設計パラメータ探索を行った．Zynq 評価ボードである ZCU102 を用いた評価の結果，ソフトウェア実行に比べフレームレートを最大で 4.82 倍向上可能であることが分かった．

1. はじめに

自動運転やバーチャルリアリティなど様々な分野で自動車やロボットなどの機器が自律動作するニーズが高まっている．自律動作に必要な周囲環境の認識方法としてカメラ画像を用いる画像認識がある．中でも，自己位置を推定するための手法として SLAM (Simultaneous Location and Mapping) [6] の一種である VSLAM (Visual SLAM) [7] が用いられている．自律動作の実現にはリアルタイム処理が必要であるため，高速化が求められている．VSLAM の大部分を占める画像中の特徴点に関する処理をハードウェア実装により高速化する研究が行われている [1][3][8].

本稿では VSLAM 実装の 1 つである ATAM [16] にハードウェア–ソフトウェアパーティショニング [17] を適用することにより性能最大化を目指す．これは，アプリケーションを汎用プロセッサを用いたソフトウェア処理と専用プロセッサを用いたハードウェア処理に分割実装する手法である．このようなハードウェア–ソフトウェアハイブリッドシステムでは，コストと性能間の良好なトレードオフポイントを発見が重要である．アプリケーション性能への影響が大きい機能はハードウェア実装し，そうではない機能はソフトウェア実装することが望ましい．

一般的に，アプリケーションは複数の機能から構成される．複数機能を回路実装する場合，回路面積制約およびタイミング制約を満たすか否かは配置配線後に判明する．そのため，最適な回路構成の探索は時間的コストが高く困難である．そこで，それぞれの機能の資源量と性能のトレードオフをパラメータ化可能な IP 生成法を用いて設計空間を削減し，現実的な時間での設計空間探索を可能とする．

そのような IP を生成するために，画像処理を効率よく記述するための DSL (Domain Specific Language) である Halide [14] プログラムから高位合成可能なプログラムを生成する Halide FPGA バックエンドを用いた．Halide はアルゴリズム部分とスケジューリング部分を分けて記述する．アルゴリズム部分には計算内容を記述し，スケジューリング部分には処理の順序や並列度などを記述する．Halide FPGA バックエンドを用いることで，スケジューリング部分で指定する並列度をパラメータとみなすことで，トレードオフを考慮した IP を生成できる．

Zynq 評価ボードである ZCU102 向けに ATAM をハードウェア–ソフトウェアハイブリッドシステムとして実装した．性能評価の結果，ソフトウェア実装と比較しフレーム

[1] 九州大学
[2] 大分工業高等専門学校
[3] 株式会社フィックスターズ
[4] Fixstars Solutions Inc.
[5] 産業技術大学院大学
[a)] ryoji.hara@cpc.ait.kyushu-u.ac.jp
[b)] tteruo@kyudai.jp
[c)] inoue@ait.kyushu-u.ac.jp

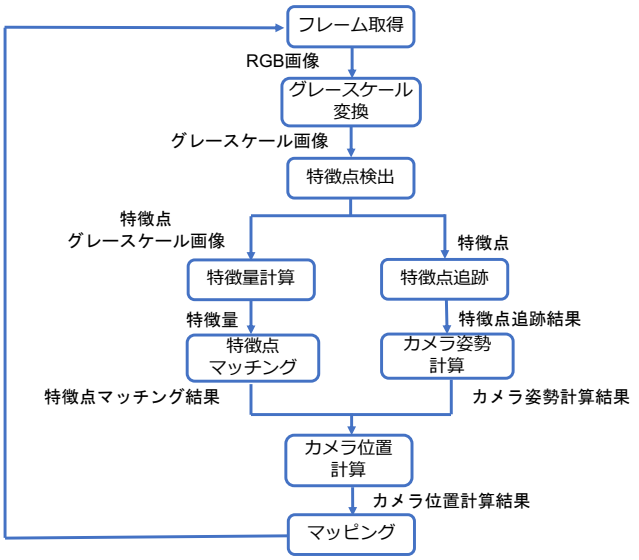

図 1　簡略化した ATAM の処理フロー

レートが最大で 4.82 倍向上可能であることが分かった．

本稿の主な貢献点は以下の 3 点である．

1. ATAM の処理時間解析
2. 主要機能の Halide 実装による ATAM のハードウェア–ソフトウェアハイブリッドシステム実装
3. ハードウェア–ソフトウェアパーティショニングした ATAM の性能評価

本稿の構成は以下の通りである．第 2 章では本研究でターゲットとした VSLAM アプリケーションの 1 つである ATAM について述べる．第 3 章で本研究で用いたパラメータ化された IP の生成手法について述べ，第 4 章でハードウェア-ソフトウェアハイブリッドシステムについて述べる．第 5 章で本研究でハードウェア実装した機能について述べ，第 6 章でハードウェア実装した機能の単体評価と，ATAM のアプリケーション性能評価を行う．第 7 章で関連研究について説明し，第 8 章で本研究をまとめる．

2. VSLAM アプリケーション：ATAM

ATAM は画像処理ライブラリ OpenCV を用いた VSLAM の C++実装の 1 つである．ATAM の処理フローを図 1 に示す．ATAM はフレーム取得，グレースケール変換，特徴点検出，特徴点追跡，カメラ姿勢計算，特徴量計算，特徴点マッチング，カメラ位置計算，マッピングで構成される．

ATAM の処理フローを説明する．まず，フレーム取得によりカメラから RGB 画像が取得され，次にグレースケール変換によりグレースケール画像に変換される．次に，特徴点検出により画像中の特徴点を検出する．特徴点検出後の処理は特徴点追跡と特徴量計算の 2 つに分かれる．特徴点追跡により，検出された特徴点の前フレームからの移動量を算出する．特徴点追跡の結果はカメラ姿勢計算に用いられる．カメラ姿勢計算ではカメラの向きを算出する．特

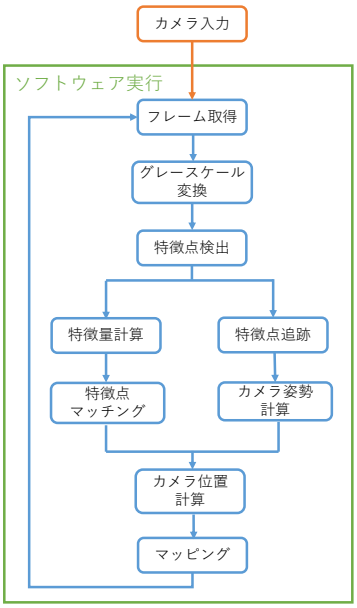

図 2　ATAM のソフトウェア実行処理フロー

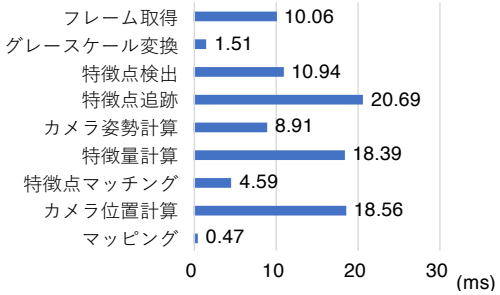

図 3　ATAM の処理時間解析結果

徴量計算では，特徴量と呼ばれる検出された特徴点それぞれの特徴を定量化した値を算出する．この値は特徴点マッチングに用いられる．特徴点マッチングでは現在のフレームで算出された特徴量とキーフレームとして登録されているフレームの特徴量が比較され，これら 2 フレーム間で一致する特徴点のペアを作成する．特徴点マッチングとカメラ姿勢計算の結果はカメラ位置計算に入力され，現在のカメラ位置を算出する．その後，地図を作成する．

ATAM の性能に対する影響の大きい機能を適切に選択するため，ATAM の処理時間を解析した．ATAM のハードウェア–ソフトウェアハイブリッドシステムと評価条件を統一するため，カメラ画像の取得に最低限必要な機能には FPGA を用いた．その際の処理フローを図 2 に示す．橙枠はハードウェア実行を示し，青枠はソフトウェア実行を示している．カメラから信号が届き次第，後続の機能がソフトウェア実行される．処理時間解析結果を図 3 に示す．特徴点に関する機能である特徴点検出，特徴点追跡，特徴量計算，特徴点マッチングの処理時間が比較的長く，それぞれ 1 フレームの処理時間の 11.63%, 21.99%, 19.54%, そして 4.88%を占めている．したがって，これらの機能をハードウェアで実装し ATAM の高速化を図る．

```
1  Var x,y;
2  Func f;
3
4  // Algorithm description
5  f(x, y) = in(x-1, y) + in(x, y) + in(x+1, y);
6
7  // Scheduling description
8  f.parallel(y, 4).vectorize(x);
```

図 4　Halide での 1 次元畳み込み処理実装例

3. Halide を用いた FPGA 実装

3.1 画像処理向け DSL：Halide

Halide [14] は画像処理やテンソル計算に特化した DSL (Domain Specific Language) である．Halide では，図 4 のようにアルゴリズム部分とスケジューリング部分を分離して実装する．アルゴリズム部分では純粋関数型で計算の意味のみを記述する．プログラム中に可算ループ以外の制御構造が出現しないため，コントロールフローの解析が不要である．さらに，再帰呼び出しが禁止されているため副作用やループ伝搬依存ないことが保証され，ループ依存解析なしに様々なループ変形を適用できる．このように，ドメイン特化によって，コンパイラによる簡易解析で積極的な最適化を行えるように言語仕様が設計されている．

スケジューリング部分では，アルゴリズム記述の計算の意味を保ったまま，計算の順序やデータの保持方法の指定を行う．このように最適化のための記述がアルゴリズムの記述と分離されることによって，C/C++などの手続き型言語での高速化において発生するソースコードの難読化や，ソースコードがターゲットハードウェア毎にマルチバージョン化される問題を解決できる．そのため，マルチターゲットハードウェア向けの最適化を容易に実現できる．

3.2 Halide FPGA バックエンド

Halide FPGA バックエンドは株式会社フィックスターズによって開発中の Halide コンパイラバックエンドであり，Halide で記述されたプログラムを Xilinx 社の FPGA 向け高位合成処理系である Vivado HLS [18] 向けに最適化された C++記述に変換する．

3.2.1 生成ハードウェアアーキテクチャ

Halide FPGA バックエンドは，関数の定義式における各演算をノードとしたデータフローアーキテクチャを生成する．Halide のアルゴリズムは純粋関数として記述されるため，比較的容易にデータフローアーキテクチャに変換可能である．また，各演算を 1 サイクルごとに命令実行できるようパイプライン化を試みる．

FPGA バックエンドが生成するハードウェアアーキテクチャを図 5 に示す．ルートレベルで計算される Func の計算をモジュールの単位とし，モジュール間もパイプライン化する．モジュール間のバッファリングの手法として，メモリ I/O とストリーム I/O の 2 種類をサポートする．

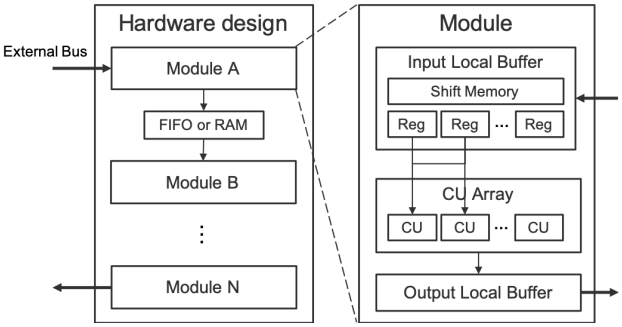

図 5　Halide FPGA バックエンドが生成するハードウェアアーキテクチャ

メモリ I/O では，FPGA 内の SRAM あるいは FPGA 外の DRAM 上にデータを配置し，アドレッシング可能なメモリバスを介してアクセスする．データに自由にアクセスできるため柔軟性は高いが，以下の理由で性能やハードウェアリソースの使用量の観点では効率が良いとは言えない．

- SRAM の場合バッファ全領域分のメモリ量が必要
- DRAM の場合データ転送が性能ボトルネックになる
- 前段のモジュールでの全ての要素の書き込みが終わるまで後続のモジュールでの読み込みができない

ストリーム I/O では，FPGA 内の SRAM や LUT (Lookup Table)，FF (Flip-Flop) などによって構成された FIFO 上にデータを配置する．モジュール間でのデータの依存関係が明確であるため，モジュール間を跨ってパイプライン化できる．そのため，少ないハードウェアリソースで性能効率の良い処理を実現可能となる．一方で，読み込み側は FIFO の先頭，書き込み側は FIFO の末尾にしかアクセスできないため，メモリ I/O に比べ実行可能な処理の柔軟性は低い．

Halide が対象とする画像処理やテンソル計算では，注目要素の周辺を参照するステンシル計算が必要となる場合が多い．そこで，ストリーム I/O で周辺画素を参照する際は，ラインバッファを生成する．プログラマがストリーム I/O の適用可否の判断やラインバッファの必要容量の計算をするのは難しいため，後述のコンパイルフローにおけるアクセスパターン解析と参照範囲解析において，ストリーム化判定とラインバッファの容量を決定する．

このように，Halide FPGA バックエンドは入力された Halide プログラムのアルゴリズムに特化したハードウェアを生成し，メモリや制御用の回路などのハードウェア資源の削減を試みる．更に内部は可能な限りパイプライン実行されるため，演算回路の利用効率も向上できる．また，Halide のアルゴリズム記述から専用ハードウェアを自動生成可能なため，専用アーキテクチャ生成におけるアルゴリズム変更時の実装コストが大きい問題を解消し，ハードウェアを短期間で実装可能である．

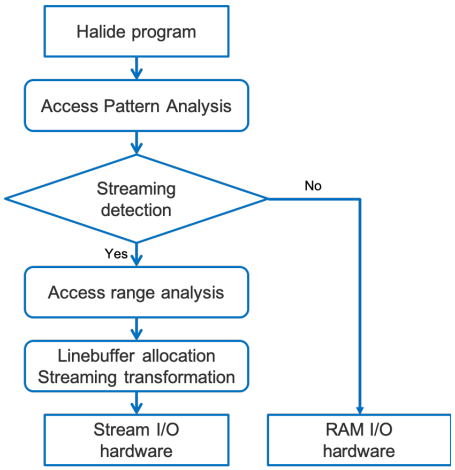

図 6　Halide FPGA バックエンドのコンパイルフロー

3.2.2　Halide FPGA バックエンドのコンパイルフロー

Halide FPGA バックエンドのコンパイルフローを図 6 に示す．Halide コンパイラは入力となる Halide プログラムのアルゴリズム記述とスケジューリング記述から，Halide-IR を生成する．Halide-IR はアルゴリズム記述には出現しないループや条件分岐などの制御文が追加されており，関数の入出力サイズの定義域に対して，スケジューリング記述に対応する計算オーダーを実現する計算ループを構築する．Halide FPGA バックエンドでは，コンパイラバックエンドに非依存なフローに加えて下記の解析と最適化を行う．

アクセスパターン解析　プログラム中の各バッファアクセスにおける index 式についての LMAD (Linear Memory Access Descriptor) [13] を作成する．全てのアクセスパターンが LMAD が表現でき，かつアクセス間隔が 1 以上であるバッファに対してストリーム変換可能と判定し，それ以外のバッファはメモリ I/O に変換される．

参照範囲解析　ストリーム判定されたバッファアクセスに対して，ラインバッファの必要容量を決定する．具体的には，index 式とループ変数の上限値やそのバッファアクセスを行うための条件式などの制約を入力とし，SMT ソルバを用いて最大値を求める．

ストリーム変換　ストリーム I/O とラインバッファ間のデータ操作や，バッファのシフト処理を挿入する．

3.2.3　Halide FPGA バックエンドでのスケジューリング

FPGA バックエンドでは，他バックエンドと同様に異なる性能パラメータの調整や，ハードウェアリソースの使用量とのトレードオフを調整可能である．以下に FPGA バックエンドで主に使用されるスケジューリングを示す．

`unroll`　既存の Halide に存在するスケジューリング関数であり，FPGA バックエンドではパイプライン内における各演算器を指定数分並列に展開する．ハードウェアリソースを消費する代わりに，1 サイクルで複数のデータを同時に計算できるようになるため，計算のスループットを向上させることが出来る．

`hls_burst`　FPGA バックエンド専用のスケジューリング関数であり，バッファにおける 1 アクセス単位の要素数を指定する．FIFO や SRAM は同一サイクルでアクセス出来る回数が限られているため，演算器だけを並列に展開しても I/O がボトルネックになってしまう．本スケジューリングを適用すると単一アクセスで連続した複数の要素にアクセス出来るようになるため，I/O のスループットを向上可能である．

`hls_interface`　FPGA バックエンド専用のスケジューリング関数であり，各バッファに割り当てるハードウェアリソースを指定する．また，IP のインターフェースに使用するプロトコルも指定できる．

性能と消費リソースの制約や I/O のインターフェースはシステム要件によって決定されるため，最適であるものを自動決定することは難しい．しかしながら，要件に依存するアーキテクチャパラメータをスケジューリング関数によりユーザ指定可能にすることで，アーキテクチャの自動生成による省力化と設計自由度の両立を実現している．

4.　ハードウェア–ソフトウェアハイブリッドシステム

4.1　ハイブリッドシステムの概要

ハードウェア–ソフトウェアハイブリッドシステムは，アプリケーションを汎用プロセッサを用いたソフトウェア処理と専用のハードウェア処理に分割するハードウェア–ソフトウェアパーティショニング [17] を適用して構成されたシステムである．本稿では，汎用プロセッサと FPGA を搭載した SoC である Zynq 評価ボードをターゲットデバイスとする．回路規模の問題からアプリケーション全体を FPGA 実装できない前提で，FPGA 実行に向き，かつ性能への影響が大きい機能を FPGA に実装し，その他は汎用プロセッサでソフトウェア処理するシステムを考える．

このようなシステムでは，ハードウェア実行–ソフトウェア実行間のインタフェースが必要となる．また，ハードウェア化部分が複数ある場合は，それぞれの機能の資源量と性能のトレードオフをとりつつターゲットとする FPGA に収まる構成を探索する必要がある．次節でハードウェア–ソフトウェア間のインタフェースについて延べ，4.3 節でハードウェア構成のパラメータ探索法について述べる．

4.2　ハードウェア–ソフトウェア間インタフェース

ソフトウェアから FPGA IP に実装した機能を呼び出し，また，データの入出力を行う必要がある．FPGA IP を呼び出すためには，制御用のドライバソフトウェアが必要となる．これは OS カーネルに含まれるデバイスドライバを呼び出し，FPGA IP を制御する．本稿では，OS として Linux

を使用し，汎用のデバイスドライバである uio (user-space io) を用いた．uio を用いる場合，FPGA IP のレジスタ空間は I/O メモリ空間にマップされ，そのメモリ空間をユーザ空間で動作するドライバソフトウェアでアクセスし FPGA IP を制御できる．

データの入出力は処理に応じたインタフェースを用いて行う．Halide FPGA バックエンドは Halide で記述された関数を FPGA に変換する際に，状況に応じてデータの配置に FIFO を用いるストリーム I/O またはランダムアクセスバッファを用いるメモリ I/O デザインへと変換する．ストリームは一方向に流れるデータの系列であり，高性能でありハードウェアリソース消費量も少ない．ランダムアクセスバッファはストリームとして変換できない場合に用いられ，ハードウェアリソースを多く消費し，性能も低くなる．

ストリーム I/O を用いる場合, FPGA IP と汎用プロセッサのメモリとの間のデータの移動には DMA (Direct Memory Access) を用いる．FPGA IP への入力，IP からの出力の両方で DMA を用いる．従って，ストリーム I/O を用いる場合には，FPGA IP の他に少なくとも 2 つの DMA を制御する必要がある．汎用プロセッサ側のメモリ（ホストメモリ）にデータを書き込み，入出力両方の DMA および FPGA IP を起動すると，順次データが FPGA IP に DMA により送られ処理される．処理されたデータも DMA により，ホストメモリに書き込まれる．全てのデータが転送されると，処理は終了する．DMA の制御にも， FPGA IP と同様にデバイスドライバには汎用のデバイスドライバである uio を用い，マップされた I/O メモリ空間を DMA のドライバソフトウェアでアクセスし制御する．

一方，メモリ I/O を用いる場合は FPGA IP と汎用プロセッサ間で共有するメモリを介してデータをやり取りするため DMA は不要である．汎用プロセッサで動作するソフトウェア処理により, FPGA からアクセス可能な共有メモリ空間にデータを書き込み，FPGA IP を起動し，FPGA IP の処理修了後に共有メモリから処理後のデータを取得する．

4.3 IP のパラメータ探索法

Halide を入力としたハードウェア生成の強みの 1 つはスケジューリング記述のみの変更により IP 内の演算の並列度および IP の I/O のデータ幅を変更可能な点である．FPGA IP は第 3.2.3 節に示すスケジューリングにより計算の順序やデータの保持方法が決定される．これにより IP が消費するハードウェア資源量や処理時間が変化する．

IP 内部の処理が複数の処理（ここでの処理とは，Halide 表現における Func で定義され，スケジューリングによってその結果が評価されるもの）により構成される場合，それぞれの処理に対して並列度を与える必要があり，これらは独立したパラメータと考えることができる．そのため，1 つの IP が持つパラメータの空間は処理ごとのパラメータの取りうる値の数を乗じたものとなる．

さらに，アプリケーションを構成する複数の機能をそれぞれ IP として生成し，それを利用可能な回路資源内に配置することを想定する．このとき，アプリケーション全体の設計空間は各 IP のパラメータの通り数を乗じたものとなり，膨大な通り数になりうる．そこで，本研究では IP のパラメータ探索を単一の IP を対象としたパラメータ探索と，それらを同時に配置するためのアプリケーション全体のパラメータ探索の 2 段階に分ける．

4.3.1 単一 IP のパラメータ探索

単一 IP については，Halide 表現を作成した時点でスケジューリング記述に与えるパラメータ空間を設定する．そして，空間に含まれるすべてのパラメータセットに対して HLC コンパイラによる高位合成用表現の作成，高位合成ツールによる IP 生成，論理合成及び配置配線ツールによるハードウェア生成を行う．さらに，ハードウェア生成の結果，ハードウェア資源量及びタイミングに関する制約を満たしたすべてのハードウェアそれぞれを実際にターゲットデバイスである評価ボードを用いて実行時間を計測する．

これにより，パラメータごとの実行時間と消費ハードウェア資源量を得ることができる．消費ハードウェア資源量は，FPGA をターゲットデバイスとする場合，LUT や FF, BRAM, DSP などの複数種類のリソースに分かれるため，パラメータの評価結果は多次元の情報を持つ．そして，この多次元の情報に対してパレート解となるようなパラメータの抽出を行う．抽出されたパラメータ群は実行時間とハードウェア資源量のトレードオフを持つため，これらを IP のパラメータ候補とする．

4.3.2 アプリケーション全体のパラメータ探索

アプリケーション全体については，単一 IP におけるパラメータ探索により得られたパラメータ候補を用いたすべての組み合わせについて探索を行う．具体的には，論理合成及び配置配線ツールによるハードウェア生成を行い，制約を満たすものについてアプリケーション性能の評価を行い，最適なパラメータを決定する．

この探索の重要な利点は，個々の機能（IP）の性能向上率ではなく，アプリケーション性能に対する寄与と機能に対して割り当てる資源量が最適化されることである．これは，アプリケーションを構成するそれぞれの IP をパラメータを持たせて実装する必要がある．従来のハードウェア設計では，このパラメータを持たせることはできても，それを変更する際に必要な記述の変更量が大きかったり，または，処理内容と記述が混在しているため変更可否の判断が困難であった．アプリケーション性能を指標として用いるパラメータ探索は，Halide 記述により処理内容と並列度を分離可能であるからこそ実現可能であると考えられる．

表 1　OpenCV を用いた特徴点マッチングで使用可能な距離計算

FlannBased	FLANN メソッド [11]．L2 ノルムを使用．
BruteForce	総当たりメソッド．L2 ノルムを使用．
BruteForce–SL2	総当たりメソッド．二乗 L2 ノルムを使用．
BruteForce–L1	総当たりメソッド．L1 ノルムを使用．
BruteForce–Hamming	総当たりメソッド．ハミング距離を使用．

5.　ATAM のハイブリッドシステム実装

5.1　ハードウェア実装機能

第 2 章の処理時間解析に基づき，本稿では特徴点検出 [19]，特徴量計算，特徴点マッチング，特徴点追跡の 4 つの機能を専用ハードウェア実装する．本章では，特徴量計算，特徴点マッチング，特徴点追跡について説明する．

5.1.1　特徴量計算

ATAM では ORB 特徴量 [15] を用いる．これは BRIEF 特徴量 [4] を基にしており，方向の計算を拡張したものである．まず，特徴点を中心とする画像パッチに対し輝度値で重み付けした重心を求める．求めた重心から画像パッチの中心である特徴点までのベクトルの方向が ORB 特徴量の方向となる．方向は以下の流れで計算される．

1. 0,1 次モーメント m_{pq} を輝度値 $I(x,y)$ により求める．
$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{1}$$
2. 重心 C を求める．
$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2}$$
3. C と画像パッチの中心を結ぶ直線の傾きを求める．
$$\theta = arctan2(m_{01}, m_{10}) \tag{3}$$

次に，この方向に対して BRIEF 特徴量を計算する．BRIEF 特徴量は以下の式で計算される．

$$\begin{cases} f(u,v) = 1 & (I(u) - I(v) \geq 0) \\ f(u,v) = 0 & (I(u) - I(v) < 0) \end{cases} \tag{4}$$

ここで，$f(u,v)$ は BRIEF 特徴量，$I(u)$ は輝度値，u,v はランダムにサンプルした 2 点のペアである．1 ペアにつき特徴量の 1 ビットが割り当てられるため，ペア数が特徴量のビット数となる．本研究では特徴量を 256 ビットとしているため，256 組のペアを用いて BRIEF 特徴量を計算した．

5.1.2　特徴点マッチング

特徴量はある画像の特徴点と他の画像の特徴点との一致を判断する際に用いられる．2 つの特徴点の特徴量は距離計算により比較される．その際，ある特徴点は最も距離の短い特徴点と一致すると見なされる．OpenCV を用いた特徴点マッチングで使用可能な距離計算法を表 1 に示す．

ATAM でのマッチングには BruteForce–Hamming が用いられている．BruteForce–Hamming では，ある画像 A の特徴点の特徴量とある画像 B の特徴点すべての特徴量をハミング距離で比較し，最短距離の組を同一と判定する．

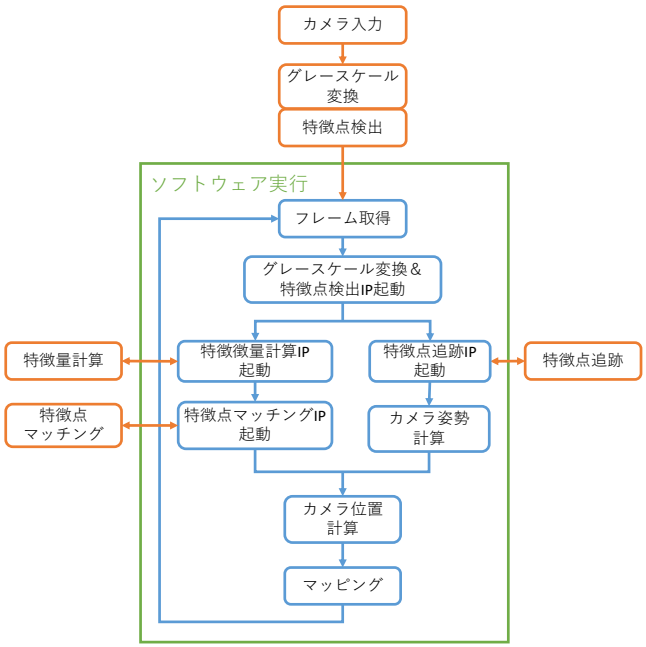

図 7　ATAM のハードウェア–ソフトウェアハイブリッドシステムの処理フロー

5.1.3　特徴点追跡

ATAM では特徴点追跡にオプティカルフローを用いる．これは，2 枚の画像を比較し，画素の移動量を計算するアルゴリズムである．本稿では，Lucas–Kanade 法によるオプティカルフローを実装した．この手法は，山登り法によって計算精度を高めるため，画像のピラミッド階層を作成し，各階層において移動量の近似解を逐次的に求める．

ピラミッドの層数，および逐次処理の回数は，出力精度に影響を及ぼすだけでなく，ハードウェア実装に必要なリソース消費量に影響を及ぼすパラメータである．そこで，これらの設計パラメータを変更して実験を行い，精度を確認した上でパラメータを決定した．なお，これらは本稿で探索する性能と回路資源に関わるパラメータとは異なり，精度に影響する点で質的に異なるパラメータである．

オプティカルフローの出力誤差を定量的に計測するため，移動距離が判明している 2 枚 1 組の画像を用意し，Halide 実装とソフトウェア実装それぞれの出力結果を正解と比較した．その結果，階層数が 5, 逐次処理回数が 1 であれば x 方向，y 方向とも ±15 画素程度の移動距離内なら絶対誤差 0.14 以内で解を出力することが分かった．これは，許容可能な誤差であると判断し，このパラメータ値を採用した．

5.2　ハイブリッドシステム実装

図 7 に ATAM ハイブリッドシステムの処理フローを示す．橙枠はハードウェア実行，青枠はソフトウェア実行を示す．特徴点検出機能はソフトウェアによる制御を必要としない．特徴量計算，特徴点マッチング，特徴点追跡機能はソフトウェアにより制御され，DMA を介してデータのやりとりを行う．

表 2　評価に用いたハードウェア環境 (ZCU102)

CPU	Cortex®-A53 64-bit quad-core processor
LUT	274,080
FF	548,160
DSP	2,520
BRAM	1,824 (32.1 MB)
メモリ	DDR4 4 GB

表 3　評価に用いたソフトウェア環境

OS	Linux 4.14.0
C++コンパイラ	g++ 7.3.0
HLC	1.3.1 および 2.5.0
Vivado	2018.2
Vivado HLS	2018.2

6. 評価

6.1 評価概要

本稿では，ATAM にハードウェア–ソフトウェアパーティショニングを適用しアプリケーション性能を評価する．まず，ハードウェアで実装した機能のうち，特徴点検出機能 [19] に加え新たに特徴量計算，特徴点マッチング機能の単体評価を行った．また，4 つの機能を同時に FPGA 実装したハイブリッドシステムのフレームレートをソフトウェア実行と比較評価した．

6.2 評価環境

評価に用いたハードウェア環境を表 2 に，ソフトウェア環境を表 3 に示す．実装するソフトウェア–ハードウェアハイブリッドシステムはカメラを FPGA に接続する．カメラ入力のフレームレートは 30 fps であり，カメラから入力したすべてのフレームを処理可能とすること，すなわち 30 fps 以上での処理が本稿における設計目標である．

6.3 ハードウェア実装機能単体評価

ORB 特徴量計算および特徴点マッチングを実行する IP を実行する最低限のハードウェアを生成し，それらの実行時間計測を行った．単体評価の目的は，複数のパラメータに対して回路資源使用量および実行時間を調べ，パレート最適となるパラメータを探索することである．また，オプティカルフローを実行する IP については時間の都合上 1 パラメータのみとし，パラメータ探索は行わない．

本来は LUT や FF，DSP などの資源量も考慮する必要があるが，本稿では BRAM 資源の制約が比較的厳しいため，BRAM 使用量のみに着目しパレート解となるパラメータを探索した．ORB 特徴量計算を実行する IP のパラメータ探索結果を図 8 に，特徴点マッチングを実行する IP のパラメータ探索結果を図 9 に示す．ORB 特徴量計算を実行する IP のパレート解となるパラメータは矢印と数字で示

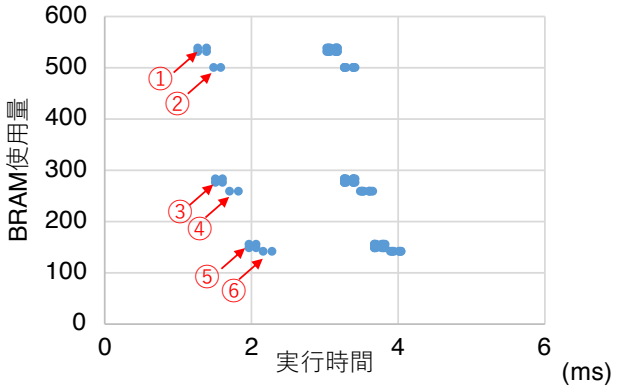

図 8　BRAM に着目したパレート解となるパラメータ探索（ORB 特徴量計算）

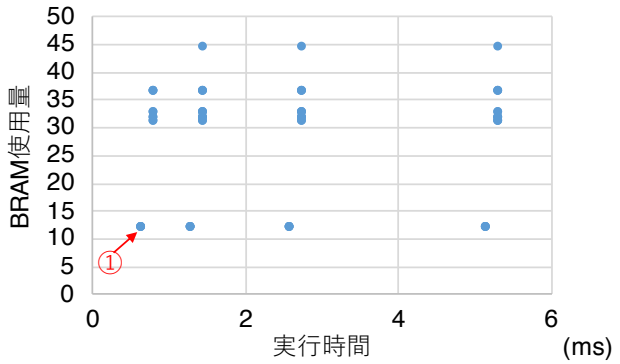

図 9　BRAM に着目したパレート解となるパラメータ探索（特徴点マッチング）

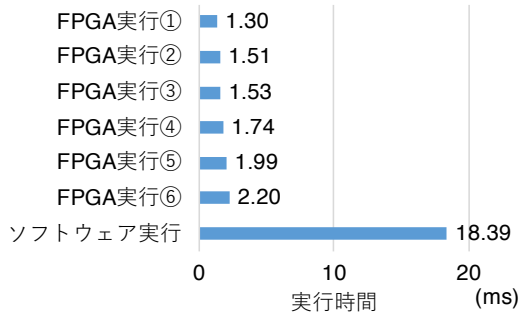

図 10　ORB 特徴量計算実行時間評価結果

している 6 つである．特徴点マッチングを実行する IP のパレート最適となるパラメータは 1 つのみである．

パレート解となる ORB 特徴量計算 IP の実行時間，CPU での OpenCV が備える ORB 特徴量計算プログラムの実行時間を図 10 に示す．パレート最適となる ORB 特徴量計算 IP での実行時間は CPU での OpenCV が備える ORB 特徴量計算の実行時間より最大で 94.4%削減される．

パレート解であるパラメータによる特徴点マッチング IP の実行時間，CPU での OpenCV が備える特徴点マッチングプログラムの実行時間を図 11 に示す．パラメータ探索を適用した FPGA 実行によりソフトウェア実行に比べ実行時間が 86.6%削減された．

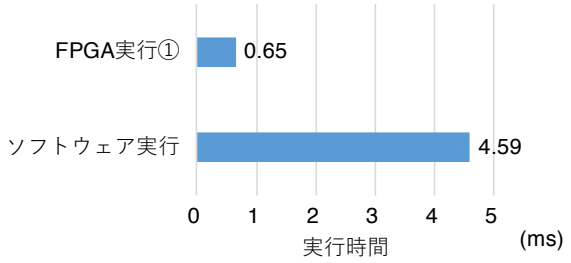

図 **11**　特徴点マッチング実行時間評価結果

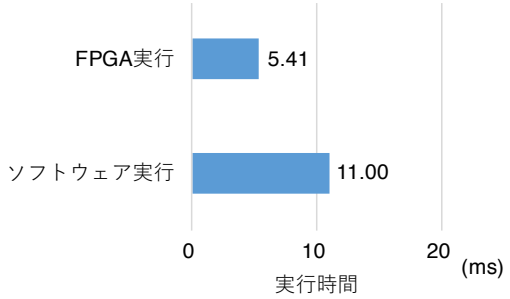

図 **12**　オプティカルフロー実行時間評価結果

表 **4**　実質的な 1 フレームの処理時間

実行方法	実行時間 (ms)
特徴量計算 IP ④ 使用	19.51
特徴量計算 IP ⑤ 使用	19.72
特徴量計算 IP ⑥ 使用	21.37
ソフトウェア実行	94.11

オプティカルフロー IP の実行時間，CPU での OpenCV が備える特徴点マッチングプログラムの実行時間を図 12 に示す．オプティカルフロー IP での実行時間は CPU での OpenCV が備えるオプティカルフローの実行時間より 51.8%削減された．

6.4　ATAM のフレームレート評価

単体評価で得られたパレート解を用いて，ATAM の性能に対する最適な IP の組み合わせを探索した．特徴量計算 IP では 6 つ，特徴点マッチング IP については 1 つのパレート解を用い，オプティカルフロー IP については時間の都合上 1 パラメータのみを用いた．特徴量計算 IP の 6 つのパレート解のうち，他の IP と共に FPGA に実装可能なものは，④，⑤，⑥の 3 つであった．

ハードウェア生成可能な実装全てにおいて，目標である 30 fps を達成した．処理時間がフレーム取得間隔よりも短い場合，フレーム取得待ち時間を調べることでカメラ入力のフレームレートの制約を無視した場合に達成可能なフレームレートを推定できる．すなわち，1 フレームの処理時間からフレーム取得の処理時間を引いた値が実質的な 1 フレームの処理時間である．実装可能な④，⑤，⑥の IP それぞれを用いた場合の実質的な 1 フレームの処理時間を表 4 に示す．この結果により，ATAM の最適な実装は特徴量計算 IP ④を用いた実装であると言える．

7.　関連研究

7.1　ハードウェア-ソフトウェアパーティショニング

設計するシステムが複雑になるにしたがって，アプリケーションを構成する機能のハードウェア–ソフトウェアパーティショニングも複雑化し困難となる．そのため，可能な限りハードウェア–ソフトウェアパーティショニングの自動化を目的とした研究がなされてきた [10]. しかしながら，ハードウェア–ソフトウェアパーティショニング自動化の研究はターゲットアーキテクチャに特化しており一般化するのが難しい．それらの技術を定性的および定量的に分析するために，一般的なハードウェア–ソフトウェアハイブリッドシステム環境のための共通モデルを定式化するアプローチが提案されている [9].

また，ハードウェア–ソフトウェアパーティショニングのための新しい単純化されたモデルの提案も行われている [2]．このモデルは以下のとおりである．

- 1 つのソフトウェアコンテキスト (CPU) と 1 つのハードウェアコンテキスト (FPGA) のみを考慮．
- ソフトウェア実装コスト：実行時間．
- ハードウェア実装コスト：面積，消費電力，熱放散．
- ソフトウェア実装機能とハードウェア実装機能間に関係がある場合，通信オーバヘッドを考慮．

このモデルは分割問題の最も重要な特性のみを考慮可能とする．しかしながら，これらの手法で得られるものは近似解であり最適解ではない．本研究の手法ではボトルネックとなっている機能を探す必要はあるが，ハードウェア資源や性能のトレードオフを考慮し最適な構成を探索可能である．

7.2　パラメータ化された IP 生成手法

生産性向上のために，ハードウェア設計者は IP ライブラリから必要な機能を選択し使用する．しかしながら，この方法はアプリケーションに必要十分な回路を設計できない欠点を持つ．そこで，性能，回路面積，消費電力などのアプリケーション固有のトレードオフに合わせて調整できるパラメータ化されたデザインジェネレータによって生成される IP を使用する方法がある．

DSP アプリケーションの構成要素のひとつである離散フーリエ変換 (DFT) の IP を生成可能な手法が提案されている [12]．この DFT ジェネレータは設計者が回路の並列度を制御できるようなパラメータを持つ．この制御により設計者はハードウェア資源使用量やスループットなどの性能間のトレードオフを考慮し回路を設計できる．また，こ

のジェネレータの出力は合成可能な RTL レベルの Verilog 記述である．生成された DFT コアは，Xilinx LogiCORE library の DFT コアのパフォーマンスとコストに匹敵する．また，Xilinx library からは得られない様々な性能–コスト間のトレードオフポイントで，パラメータ化されたジェネレータを用いることにより DFT コアを生成できることが示された．しかしながら，この手法で生成可能な IP はフーリエ変換のための IP のみである．本研究の提案手法では，所望の機能を Halide で記述することによって様々な機能の IP を生成できる．

7.3 SLAM アプリケーションの高速化

ORB 特徴検出器は VSLAM で広く使われている．ORB 特徴検出器での特徴点検出とマッチングは計算量が大きいため，無人偵察機やロボットのような低消費電力の組込プラットフォームで，リアルタイム処理である ORB ベースの SLAM 実行は困難である．

そこで，ORB ベースの SLAM システムのヘテロジニアスアーキテクチャとして eSLAM が提案された [8]．eSLAM では ORB 特徴点検出器の FPGA 実装により高速化および低消費電力化した．この研究では　zynq 評価ボードである XCZ7045 が用いられた．arm プロセッサと比較して，eSLAM はフレームレートで 17.8 倍から 31 倍の処理速度向上，エネルギー効率で 14 倍から 25 倍の向上を達成した．

また，BRIEF 特徴量記述子 [5] は特徴量計算が単純であることから，BRIEF 特徴量記述子とマッチングをハードウェア実装し高速化する手法が提案された [3]．これによりピクセルストリームは最大 160 MHz で処理される．しかしながら，これらの提案手法を採用するためには，それら手法を FPGA に実装するための十分なハードウェア資源が必要である．本研究の手法ではボードのハードウェア資源を考慮し回路を設計できるため，この問題を解決できる．

また，FPGA 実装に適した新しい特徴点抽出／マッチングアルゴリズムが提案された [1]．この提案アルゴリズムはシミュレーションにより検証されている．その結果，このアルゴリズムは処理時間や精度の観点から，特徴点抽出／マッチングアルゴリズムを検討する際の新しい選択肢となるような性能を持つことが示された．しかしながら，この手法は未だ提案段階であり，また，提案手法を採用したアプリケーションのみ高速化可能である．本研究の手法は高速化したい機能の Halide 記述によりハードウェアアクセラレーション可能であり，多様な機能を高速化可能である．

8. おわりに

本稿では，ATAM にハードウェア–ソフトウェアパーティショニングを適用することにより性能最大化を目指した．具体的には，画像処理に特化した DSL である Halide から FPGA 実装する Halide FPGA バックエンドを用いて，ATAM の性能に比較的影響を与える特徴点に関する特徴点検出，特徴点追跡，特徴量計算，特徴点マッチングの機能を FPGA に実装した．Zynq 評価ボードである ZCU102 における評価の結果，ソフトウェア実行と比較しフレームレートを最大で 4.82 倍向上可能であることを明らかにした．

謝辞　本研究は一部 JST ACT-I JPMJPR18UH，ならびに，内閣府が進める「戦略的イノベーション創造プログラム（SIP）第 2 期／フィジカル空間デジタルデータ処理基盤」（管理法人：NEDO）による．

参考文献

[1] Aguilar-González, A. and Arias-Estrada, M.: Towards a Smart Camera for Monocular SLAM, *Proceedings of the 10th International Conference on Distributed Smart Camera*, ICDSC '16, New York, NY, USA, ACM, pp. 128–135 (online), DOI: 10.1145/2967413.2967441 (2016).

[2] Arato, P., Juhasz, S., Mann, Z. A., Orban, A. and Papp, D.: Hardware-software partitioning in embedded system design, *IEEE International Symposium on Intelligent Signal Processing, 2003*, pp. 197–202 (online), DOI: 10.1109/ISP.2003.1275838 (2003).

[3] Brenot, F., Piat, J. and Fillatreau, P.: FPGA Based Hardware Acceleration of a BRIEF Correlator Module for a Monocular SLAM Application, *Proceedings of the 10th International Conference on Distributed Smart Camera*, ICDSC '16, New York, NY, USA, ACM, pp. 184–189 (online), DOI: 10.1145/2967413.2967426 (2016).

[4] Calonder, M., Lepetit, V., Strecha, C. and Fua, P.: BRIEF: Binary Robust Independent Elementary Features, *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, Berlin, Heidelberg, Springer-Verlag, pp. 778–792 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1888089.1888148⟩ (2010).

[5] Calonder, M., Lepetit, V., Strecha, C. and Fua, P.: BRIEF: Binary Robust Independent Elementary Features, *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, Berlin, Heidelberg, Springer-Verlag, pp. 778–792 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1888089.1888148⟩ (2010).

[6] Durrant-whyte, H. and Bailey, T.: Simultaneous localization and mapping: Part I, Robotics & Automation Magazine, IEEE, Vol. 13, pp. 99 – 110 (online), DOI: 10.1109/MRA.2006.1638022 (2006).

[7] Fuentes-Pacheco, J., Ruiz-Ascencio, J. and Rendón-Mancha, J. M.: Visual Simultaneous Localization and Mapping: A Survey, *Artif. Intell. Rev.*, Vol. 43, No. 1, pp. 55–81 (online), DOI: 10.1007/s10462-012-9365-8 (2015).

[8] Liu, R., Yang, J., Chen, Y. and Zhao, W.: eSLAM: An Energy-Efficient Accelerator for Real-Time ORB-SLAM on FPGA Platform, *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, ACM, pp. 193:1–193:6 (online), DOI: 10.1145/3316781.3317820 (2019).

[9] López-Vallejo, M. and López, J. C.: On the Hardware-software Partitioning Problem: System Modeling and Partitioning Techniques, *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 8, No. 3, pp. 269–297 (online), DOI: 10.1145/785411.785412 (2003).

[10] Morton, A.: Hardware/Software Partitioning and Scheduling of Embedded Systems, PhD Thesis, University of Waterloo Waterloo (2005).
[11] Muja, M. and Lowe, D. G.: Fast approximate nearest neighbors with automatic algorithm configuration, *In VISAPP International Conference on Computer Vision Theory and Applications*, pp. 331–340 (2009).
[12] Nordin, G., Milder, P. A., Hoe, J. C. and Püschel, M.: Automatic Generation of Customized Discrete Fourier Transform IPs, *Proceedings of the 42Nd Annual Design Automation Conference*, DAC '05, New York, NY, USA, ACM, pp. 471–474 (online), DOI: 10.1145/1065579.1065703 (2005).
[13] Paek, Y., Hoeflinger, J. and Padua, D.: Efficient and Precise Array Access Analysis, *ACM Trans. Program. Lang. Syst.*, Vol. 24, No. 1, pp. 65–109 (online), DOI: 10.1145/509705.509708 (2002).
[14] Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F. and Amarasinghe, S.: Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines, *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, New York, NY, USA, ACM, pp. 519–530 (online), DOI: 10.1145/2491956.2462176 (2013).
[15] Rublee, E., Rabaud, V., Konolige, K. and Bradski, G.: ORB: An efficient alternative to SIFT or SURF, *2011 International Conference on Computer Vision*, pp. 2564–2571 (online), DOI: 10.1109/ICCV.2011.6126544 (2011).
[16] Uchiyama, H., Taketomi, T., Ikeda, S. and Lima, J.: [POSTER] Abecedary tracking and mapping: A toolkit for tracking competitions, *Proceedings of the 2015 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2015*, United States, Institute of Electrical and Electronics Engineers Inc., pp. 198–199 (online), DOI: 10.1109/ISMAR.2015.63 (2015).
[17] Vahid, F.: What is Hardware/Software Partitioning?, *SIGDA Newsl.*, Vol. 39, No. 6, pp. 1–1 (online), DOI: 10.1145/1862900.1862901 (2009).
[18] Xilinx: *UG902 - Vivado Design Suite User Guide: High-Level Synthesis (ver2018.2)* (2018).
[19] 原　凌司，井上優良，谷本輝夫，大澤隆志，丸岡　晃，飯塚拓郎，井上弘士：高位合成用 DSL コンパイラを用いたコーナー検出処理のハードウェア実装，情報処理学会研究報告，Vol. 2018-ARC-233 No.11, pp. 1–8 (2018 年 11 月).