一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2025-55,DC2025-127(2026-03)

PYNQを用いた可視化システムの開発支援ツールキット

濵田　海斗†　藤枝　直輝††

† 愛知工業大学大学院 工学研究科　〒470–0392 愛知県豊田市八草町八千草 1247
†† 愛知工業大学 工学部　〒470–0392 愛知県豊田市八草町八千草 1247
E-mail: †, ††{v24725vv, nfujieda}@aitech.ac.jp

あらまし　SoC (System-on-Chip) 型 FPGA (Field-Programmable Gate Array) は，画像処理や AI 推論などの高度なシステムへの利用が拡大している．こうしたシステムの開発において，内部状態や処理結果をリアルタイムに可視化することは重要である．本研究では，SoC 型 FPGA 環境である PYNQ 上で動作する可視化システムの開発支援ツールキットを提案する．本ツールキットは，高位合成により開発された図形描画ハードウェア IP と，その実装・制御を簡単にするスクリプト群から構成される．これにより，従来のソフトウェア実装よりも高速な描画性能を実現しつつ，導入の容易さを維持する．本稿では，ツールの概要と性能評価について報告する．

キーワード　FPGA (Field-Programmable Gate Array)，PYNQ，可視化

A developers' toolkit for visualization systems using PYNQ

Minato HAMADA† and Naoki FUJIEDA††

† Graduate School of Engineering, Aichi Institute of Technology
†† Faculty of Engineering, Aichi Institute of Technology
†, †† 1247 Yachigusa, Yakusa-cho, Toyota-shi, 470–0392 Japan
E-mail: †, ††{v24725vv, nfujieda}@aitech.ac.jp

Abstract　System-on-Chip (SoC) Field-Programmable Gate Arrays (FPGAs) have been increasingly used in advanced systems such as image processing and AI inference. In developing such systems, real-time visualization of internal states and processing results is crucial. We propose a development support toolkit for visualization systems running on the PYNQ SoC FPGA environment. The toolkit consists of graphics drawing hardware IP developed through high-level synthesis and a set of scripts that simplify its implementation and control. This achieves higher performance than conventional software implementation, while keeping easy deployment. This report describes the overview and performance evaluation of the proposed toolkit.

Key words　FPGA (Field-Programmable Gate Array), PYNQ, Visualization

1. はじめに

近年，画像処理や AI 推論などの高度なシステムにおいて，プロセッサと FPGA (Field-Programmable Gate Array) を同一チップ上に統合した SoC (System-on-Chip) 型 FPGA の利用が拡大している．これは，汎用プロセッサ (PS: Processing System) と FPGA ファブリック (PL: Programmable Logic) を単一チップに統合したデバイスであり，ソフトウェアの柔軟性とハードウェアの高速性を兼ね備えている．こうした高度なシステムの開発では，内部状態や処理結果をリアルタイムに可視化することがデバッグにおいて重要となる．特に，ディスプレイへの映像出力は，高解像度かつ直感的な表示が可能であり，極めて有効な手段である [1]．しかしながら，映像へのリアルタイム可視化を実現しようとすると，性能と実装コストのトレードオフに直面する．ハードウェア並みの高速性とソフトウェア並みの導入の容易性を高度に両立する可視化手法は一般的でなく，手軽に利用できる環境が不足している．

本研究の目的は，SoC 型 FPGA 開発環境において，このトレードオフを解消し，ハードウェアによる高速性とソフトウェア開発のような使いやすさを両立した，リアルタイム可視化システムを構築することである．高速性のために，高位合成 (HLS) を用いた専用描画回路を開発する．また，導入の容易性のために，回路構築とソフトウェア制御を簡略化するスクリプトとライブラリを提供する．これにより，画像処理や AI システム等の開発におけるデバッグ効率の向上と，研究開発の加速に貢献する．

本稿の構成は以下の通りである．2 節では，本研究のベースとなる先行研究を概説し，その課題を指摘する．3 節では，本

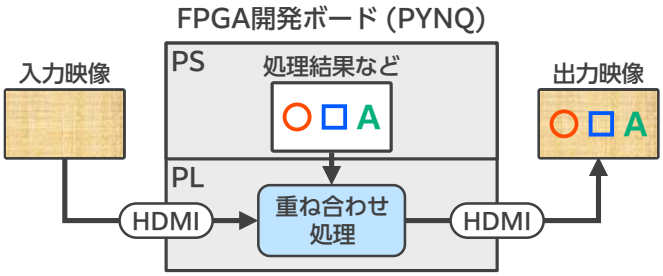

図 1　既存システムの概要

研究で提案する可視化システムの詳細を説明する．4 節で回路構築スクリプトの動作検証を行い，5 節で提案システムの性能評価を示す．最後に，6 節で本研究のまとめと今後の展望を述べる．

2. 先 行 研 究

2.1. 先行研究システムの概要

本研究室では，FPGA による画像処理応用 [2] の実演のために，HDMI 入力映像に画像をリアルタイムで重ね合わせるデモシステムを構築した．このデモは，Digilent 社製の FPGA ボードである PYNQ-Z1 を対象としており，オープンソースの PYNQ フレームワーク上で動作する．PYNQ は，PS 上の Python 環境から PL 上のハードウェア IP をドライバレスで容易に制御できるオープンソース環境である [3]．デモシステムの概要図を図 1 に示す．HDMI 入出力機能と画像の重ね合わせ機能を備えており，ボード上で実行される画像認識等の処理結果（認識枠やステータス情報など）を，入力映像に対して遅延なく重畳表示する．

2.2. 先行研究システムの課題

このデモシステムには，実用化に向けて解決すべき 4 つの主要な課題が存在する．1 つ目の課題は，ソフトウェア描画機能に起因する速度低下と描画機能の不足である．デモシステムでは，図形や文字の描画処理を PS 上の Python プログラムで行っているため，処理速度が遅い．そもそも一般的な組込みプロセッサ上での OpenCV 等によるピクセル操作は，バス帯域やキャッシュ効率の観点から HD 解像度のリアルタイム描画には不向きである [4]．特に，大量のオブジェクトを描画する際には，リアルタイム性が大きく損なわれる．加えて，システムが提供するのは単純な画素の重ね合わせ機能のみであり，直線を引く，文字を表示するといった具体的な描画機能は提供されていない．そのため，利用者はアプリケーションごとに独自に描画アルゴリズムを実装する必要があるという負担が生じる．

2 つ目の課題は，バッファリング機構の欠如に起因する表示品質の問題である．先行研究のシステムは，PL に実装された重ね合わせ表示コントローラが，BRAM (Block RAM) に保存された重ね合わせ用の描画データを常に読み出す構成になっている．このため，描画途中のデータを BRAM に書き込んだ場合，古い情報と新しい情報が同一フレーム内に混在したり，画面の表示が乱れるティアリングが発生したりする [5]．この問題は，可視化システムとしての正確性と視認性を低下させる要因となる．

3 つ目の課題は，システムの導入における複雑さの問題である．本システムを利用するためには，AMD FPGA 向けの CAD ツールである Vivado 上で，重ね合わせコントローラや HDMI 関連 IP を適切に配置・配線する必要がある．しかし，IP 数が多く手動での結線作業は煩雑であるため，導入の障壁となる．

4 つ目の課題は，PYNQ フレームワーク上での制御の複雑さの問題である．PL 上の IP を制御する際，レジスタへの直接的な値の書き込みや物理アドレスの指定など，ハードウェア特有の処理が必要となる．その結果，PYNQ フレームワークの利点である Python による手軽な制御が十分に活かされていない現状がある．

以上の課題を踏まえると，実用的な可視化環境を実現するためには，描画処理のハードウェア化による高速化と，ダブルバッファリング等による表示品質の担保が不可欠である．また，利用者の導入障壁を下げるために，複雑な回路構築や制御プロセスを抽象化し，直感的に扱える支援環境の整備も同時に求められる．

3. 提 案 手 法

2.2 項で述べた課題を解決するために，本研究ではハードウェアとソフトウェアの両面で新たな機構を取り入れた，可視化システム基盤を提案する．本システムにおいて具体的に追加する機構は，以下の 4 つである．第 1 に，3.1 項に述べる HLS による図形描画処理のハードウェア化によって，速度と機能の問題を解決する．第 2 に，3.2 項に述べるダブルバッファリング機構の導入によって，表示品質の問題を解決する．第 3 に，3.3 項に述べる回路構築自動化スクリプトの開発によって，複雑な回路構築作業を自動化し，導入の容易化を図る．第 4 に，3.4 項に述べる Python 制御ライブラリの開発によって，ユーザビリティを向上させる．

3.1. HLS による図形描画処理のハードウェア化

本項では，本システムの核心となる，高位合成 (HLS) を用いたハードウェアによる図形描画 IP の開発について述べる．開発では，ボトルネックとなっていた Python によるソフトウェア描画処理を，専用のハードウェア IP へとオフロードする．具体的には，直線・矩形・円・文字の 4 種類の基本図形を描画可能な IP を設計・実装する．これにより，不足していた図形描画機能を確立するとともに，リアルタイムな描画を実現する．

本 IP コアの設計には，AMD 社の Vitis HLS 2025.2 を使用し，C++ 言語による記述から RTL を生成する [6]．実装における最適化手法として，座標や線幅などのパラメータ計算に任意精度型を活用し，データ幅を最適化する．具体的には，通常の int 型（32 ビット）から，座標系に必要な 10-13 ビットへとデータ幅を削減し，リソース消費を削減する．また，`#pragma HLS pipeline` 等のディレクティブを使用し，描画ループ処理のパイプライン化を指示することで，処理性能の向上を図る．

本 IP コアの設計では，描画データ出力とパラメータ制御の

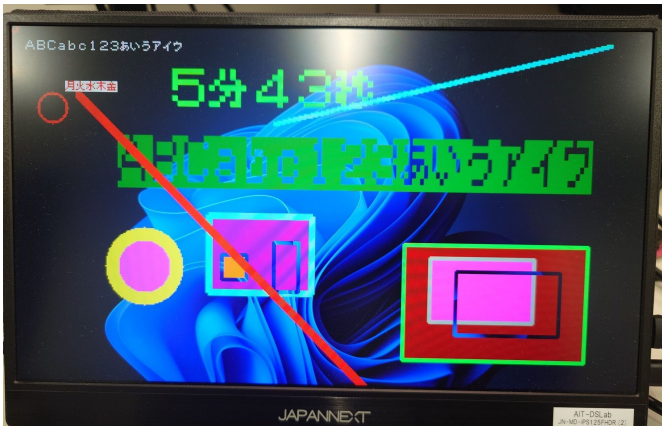

図 2　図形描画 IP のデモ映像例

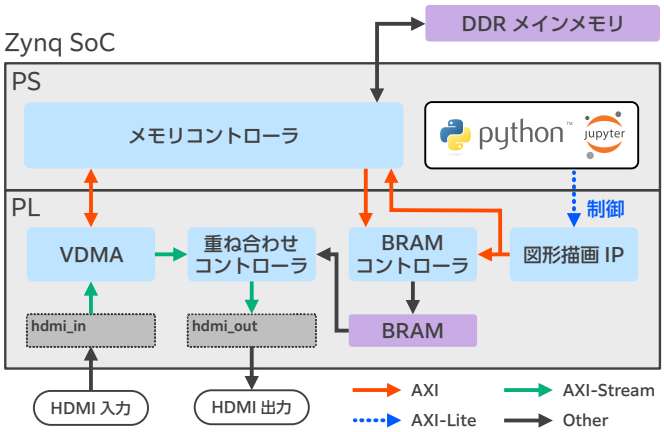

図 3　ダブルバッファリング機構を含む提案システムのデータフロー

ために 2 種類の AXI インターフェースを採用する．まず，描画データを出力するポートには，`m_axi` (AXI4 Master) インターフェースを採用する．これにより，IP が PS のメインメモリのアドレス空間へ直接データを書き込むことを可能にする．一方，座標や色などの制御パラメータを受け取るポートには，`s_axilite` (AXI4-Lite Slave) インターフェースを採用する．これにより，PS 側の Python プログラムからレジスタ経由で高速に制御可能にする．

基本図形（直線・矩形・円）については，始点や中心座標などの位置指定に加え，図形種別に応じて線幅，枠線色，塗りつぶし色をパラメータとして柔軟に指定可能である．

文字描画については，8x8 ピクセルのビットマップフォント描画機能を実装する．フォントデータには，フリーソフトウェアライセンスの「美咲フォント（美咲ゴシック第 2）」[7] を採用し，ASCII コードを拡張した独自コードにマッピングした 256 文字分のデータを IP 内部に保持する．これにより，英数字や記号のほか，ひらがな，カタカナ，一部の漢字も含む．描画処理においては，左上の頂点座標により描画位置を決定し，描画対象は文字コードによって指定する．また，X・Y 方向それぞれの拡大率や，文字色・背景色を個別に指定可能とする．

図 2 に，図形描画 IP を用いて各種図形や文字を描画し，HDMI 出力した例を示す．図から，直線，矩形，円，および文字列が描画されていることがわかる．これにより，処理結果等の多様な可視化が可能となることを確認した．

3.2.　ダブルバッファリング機構の導入

2.2 項で述べた表示品質の課題を解決するために，本システムでは，メモリ構成を刷新しダブルバッファリング機構を導入する [5]．図 3 にダブルバッファリング機構を含めた，本提案システムのデータフローを示す．本機構では，図形描画 IP の出力先を，BRAM に加え PS 側の DDR メインメモリも設定できるようにする．これにより，大容量のメインメモリ上に描画作業用のバックバッファを確保する．その結果，ユーザは描画出力先を BRAM と DDR のいずれかから選択できるようになる．

また，描画内容を反映させるための転送メカニズムを新たに実装する．描画出力先を DDR に設定した場合，描画処理を実行した段階では，DDR 上のデータのみが更新される．その後，ユーザが任意のタイミングで更新処理を呼び出すことで，DDR から BRAM（フロントバッファ）へデータを転送する仕組みである．転送が完了して初めて HDMI 出力映像が切り替わるため，描画処理と表示更新を明確に分離できる．

この仕組みにより，描画途中の不完全な状態の表示を防ぐことが可能となる．すなわち，常に完成したフレームのみを表示することが可能となるため，ティアリングのない高品質な可視化を実現する．

3.3.　回路構築自動化スクリプトの開発

本項では，Vivado 上での回路設計を支援するための，Tcl 言語 [8] を用いた自動配線スクリプトの作成について述べる．本スクリプトは，開発者が作成中の既存のブロックデザインに対して，本システムの IP 群を追加・配線する機能を提供するものである．

ユーザの利用目的や既存環境に合わせて柔軟に可視化機能を導入できるよう，機能の異なる 3 種類のシステム構成を定義する．1 つ目は，重ね合わせ表示・図形描画統合システムである．これは，先行研究の重ね合わせ表示機能に，3.1 項で設計した HLS 図形描画 IP を組み合わせた構成である．既存映像へのオーバーレイと高速な図形描画を同時に実現する．また，3.2 項で述べたとおり，図形描画 IP の出力先は BRAM と DDR のいずれかを選択可能である．これにより，直接 BRAM へ描画する単純重ね合わせ表示と，DDR を介したダブルバッファリング描画の両方に対応する．本システムは，本研究の核心となる図形描画機能と，先行研究の重ね合わせ表示機能を併せ持つため，本研究で提案する標準的な構成と位置づけられる．

2 つ目は，重ね合わせ表示システムである．これは，2 節で述べた先行研究システムと同等の機能を持つ構成であり，Python からのフレームバッファ書き込みによる単純な重ね合わせ機能のみを提供する．回路リソースを節約したい場合や，静的な情報の表示のみで十分な場合に利用されることを想定している．

3 つ目は，図形描画システムである．これは，先述した統合システムから重ね合わせ機能を除き，3.1 項の描画機能と映像出力機能に特化した構成である．外部入力を持たない演算回

路の可視化に適している．

開発した Tcl スクリプトは，ユーザが指定した構成タイプに基づいて回路を自動生成する機能を持つ．具体的には，選択された構成に必要な IP 群や階層ブロックを配置し，規格化されたインターフェース（AXI4-Stream 等）で自動配線を行う処理を実行する．これにより，ユーザは複雑な手動配線作業を省略でき，手軽に可視化システムを導入できる．

また，本スクリプトは既存回路への影響を最小限に抑えるよう設計されている．Zynq PS と PL 間のインターフェース（AXI ポート）において，既存の配線と競合を避けるために使用頻度の低いポート（HP3 や GP1 など）へ固定的に接続するようにしている．これにより，ユーザの既存回路の構成を変更することなく，安全に可視化機能を追加・配線することを可能にしている．

3.4. Python 制御ライブラリの開発

本項では，図形描画 IP を直感的に制御するための Python ライブラリクラス `DrawLib` の開発について述べる．本ライブラリは，複雑なレジスタ操作を抽象化し，OpenCV ライクな使いやすい API を提供することを特徴とする．これによりユーザは，メモリマップや物理アドレスを意識することなく，メソッド呼び出しのみで描画処理を利用できるようになる．

例えば，座標 (10, 10) に「Hello」という文字列を表示する場合，従来は文字コード変換や複数のレジスタ設定など 10 行以上にわたる処理が必要であった．一方で，本ライブラリを用いれば `dl.text(10, 10, "Hello")` のような直感的にわかりやすい 1 行の記述で実行可能である．また，`dl.update()` メソッドを呼び出すだけでダブルバッファの転送制御が行えるため，HW/SW 協調設計が容易となる．

4. 自動化スクリプトの検証

本節では，3.3 項で述べた回路構築自動化スクリプトの有用性を検証する．提案システムを実用的なものとするためには，既に IP が配置されている既存の回路に対して，その構成を破壊せずに可視化機能を安全に導入できる必要がある．そこで，用途の異なる既存回路に対し，スクリプトを用いたシステム構築実験を行う．

4.1. 検証条件と対象

3.3 項で述べた 3 種類のシステム構成について，それぞれの導入検証を行う．

重ね合わせ表示・図形描画統合システムと重ね合わせ表示システムの導入検証では，2 節で述べた画像処理応用回路 [2] を対象とする．この回路から，重ね合わせ表示のデモシステム部分を除去し，可視化機能を持たない状態をベース回路として用いる．

図形描画システムの導入検証では，大学のオープンキャンパスで用いられている真性乱数生成器 (TRNG) のデモシステムを対象とする．こちらも同様に，既存の映像出力機能を除去した状態をベース回路として用いており，そのブロックデザインを図 4 に示す．

これらの回路に対し，それぞれのシステム構成を導入するためのスクリプトを実行し，導入結果を評価する．なお，評価環境には，提案システムの PYNQ v3.0.1 に対応した AMD Vivado 2022.1 を使用する．

4.2. 検 証 結 果

まず，重ね合わせ表示・図形描画統合システムおよび重ね合わせ表示システムの導入検証結果について述べる．両システムともに，スクリプト実行によるエラーや既存回路の破壊は発生せず，正常に可視化機能を追加できた．これにより，スクリプトが既存回路に対して安全に動作することを確認した．

次に，図形描画システムの導入検証結果について述べる．スクリプト実行後に，一部手動配線を行った状態のブロックデザインを図 5 に示す．半透明で示した部分は，スクリプト実行前から存在していたベース回路の部分であり，通常色で示した部分は，スクリプトによって追加された部分である．スクリプト実行によるエラーや既存回路の破壊は発生せず，正常に可視化機能を追加できたことが確認できる．ただし，図中の赤色で示した配線は，システムの動作に必要な配線であるものの，スクリプト実行後に接続されなかったため，手作業で配線したものである．

図形描画システムの導入においてのみ，1 つの配線処理が自動で行われなかった．この配線は，既存の演算回路出力と可視化システムの入力を結ぶデータパスである．どのデータを可視化するかを決める部分であり，これはユーザの目的に依存する．そのため，スクリプトでの自動配線ではなく，ユーザの手動配線が適切であるといえる．以上の結果から，提案スクリプトは既存回路に対して安全に動作し，ユーザが必要な部分のみを手動で補完することで，容易に可視化システムを導入できることが確認できた．

5. 性 能 評 価

5.1. 実 験 環 境

本研究で構築した可視化システムの性能評価を行うため，実験に使用したハードウェアおよびソフトウェア環境を示す．まず，実験には FPGA ボードとして Digilent 社製の PYNQ-Z1 (SoC: Zynq-7000 XC7Z020) を使用する．また，ソフトウェア環境は PYNQ v3.0.1 上で構築し，Python 3.10.4 および OpenCV 4.5.4 にて動作検証を行う．なお，評価には，3.3 項で構築した重ね合わせ表示・図形描画統合システムのビットストリームを FPGA に書き込み使用する．実験に使用した評価ボードの外観と接続構成を図 6 に示す．本実験では図に示すように，PC からの映像出力をボードの HDMI 入力に接続し，ボードの HDMI 出力からディスプレイへ映像を出力するパススルー構成にて，描画機能の検証を行う．

5.2. 比 較 手 法

本実験では，提案システムの有効性を検証するため，実装方式の異なる以下の 4 つの描画手法を比較対象とする．第 1 の手法は，本研究で開発した専用ハードウェア回路を使用する HW-BRAM (Direct BRAM Drawing) である．これは図形描画 IP から FPGA 上の BRAM へ直接データを書き込むモードであり，CPU によるデータ転送を介さないため，本システムにお

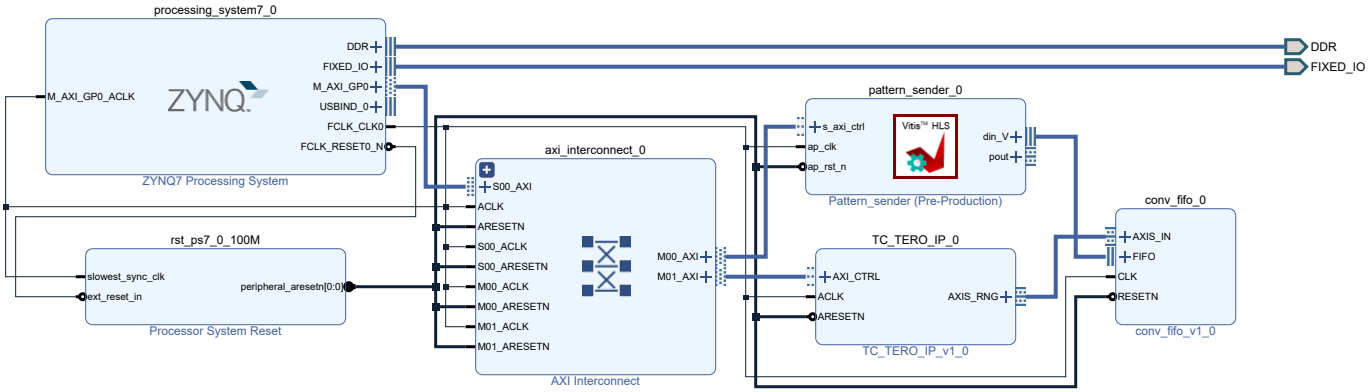

図 4　TRNG デモシステムのベース回路のブロックデザイン

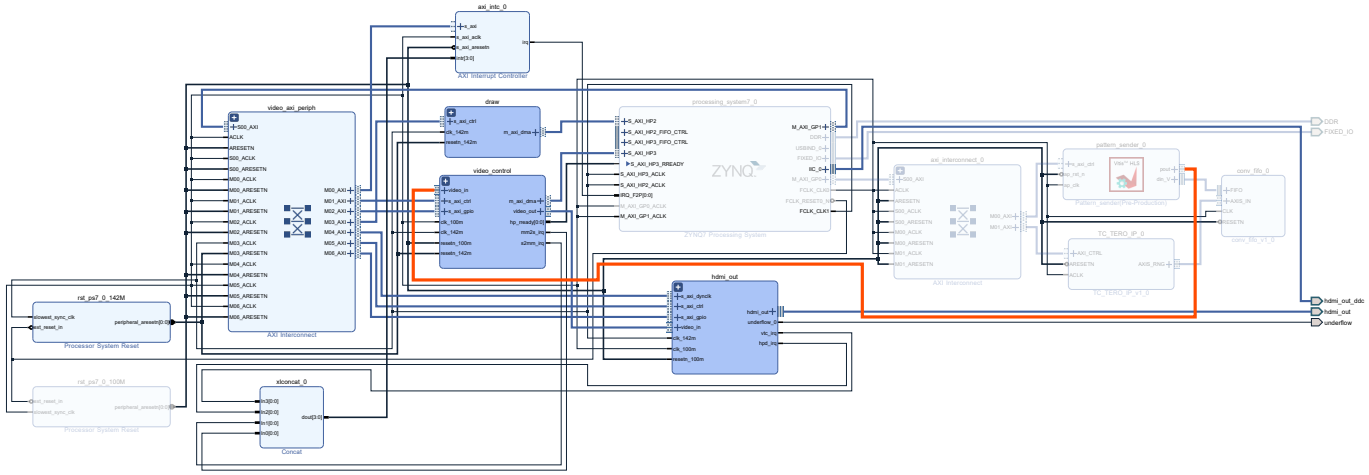

図 5　図形描画システムの自動構築スクリプト実行後のブロックデザイン

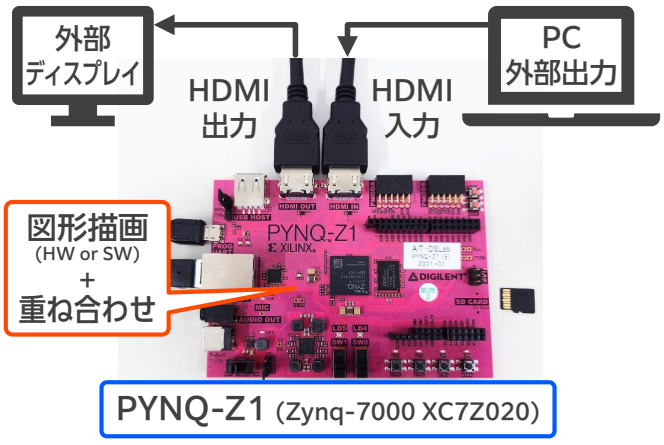

図 6　実験に使用した評価ボードと接続構成

いて最も低遅延な手法である．第 2 の手法は，同じく本研究で開発した専用ハードウェア回路を使用する HW-DB (Double Buffering) である．これは 3.2 項で述べたダブルバッファリング機構を用いるモードであり，描画過程のチラつきや古い情報の混在を防ぎ，高品質な可視化を実現するものである．第 3 の手法は，比較対象として用いる SW-CV (OpenCV) である．高速な C++ バックエンドを持つ画像処理ライブラリ OpenCV を使用し，CPU 上で描画処理を行う手法である [4], [9]．第 4 の手法は，比較の基準（ベースライン）として用いる SW-Py (Pure Python) である．これは，外部ライブラリの最適化を利用せず，Python コードのみで配列操作を行う手法であり，ハードウェア化による純粋な加速率を評価するために用いる．

5.3.　評価指標と実験条件

本実験では，提案システムの描画性能を多角的に評価するため，スループットとレイテンシという 2 つの指標を定義し計測を行う．

第 1 の指標であるスループットは，単位時間あたりの描画処理能力と定義し，動画へのオーバーレイなどフレーム単位での一括処理性能の評価に用いる．計測手順として，HW-DB およびソフトウェア実装 (SW-CV, SW-Py) においては，N 回の描画関数を連続して実行した後に画面更新関数を 1 回だけ実行し，その完了までの総時間を計測した．一方，HW-BRAM においては，描画 IP が BRAM へ直接書き込みを行い画面更新処理の必要がないため，他手法とは異なり N 回の描画関数のみを実行した時間を計測した．そして，計測した総時間を用いて $N/Time$ を計算し，1 秒間あたりに描画可能な図形数を算出した．

第 2 の指標であるレイテンシは，1 回の描画操作から画面への反映までに要する時間（単位：s）として定義し，センサ値モニタリングなどリアルタイム応答性の評価に用いた．計測手順として，HW-DB およびソフトウェア実装 (SW-CV, SW-Py) においては，1 回の描画関数と 1 回の画面更新関数をセットで

表 1　各描画対象における 10,000 回描画時のスループット値（図形数/s）

図形種別	HW-BRAM	HW-DB	SW-CV	SW-Py
直線 (Line)	1,700.30	1,642.51	22,464.24	52.62
矩形 (Rectangle)	1,463.99	1,401.62	17,376.91	4.34
円 (Circle)	1,530.66	1,636.43	21,710.17	3.41
文字 (Text)	3,086.78	2,937.10	15,685.31	521.49

実行する処理を N 回繰り返し，その総時間を試行回数 N で除算して算出した．なお，HW-BRAM においては，前述の通り画面更新処理を必要としないため，レイテンシはスループットの逆数で定義する．

また，計測実験は，直線・矩形・円・文字の 4 種類の描画対象について行う．描画対象の座標や色などのパラメータを変化させ，各手法において 10,000 回描画した際の処理時間を計測する．その際，キャッシュメモリの影響を排除するため，計測前にはウォームアップを行う．最終的に，各条件につき 5 回計測を行い，その平均値を採用する．

5.4. スループットの評価結果

各図形を 10,000 回描画した際のスループット値の一覧を表 1 に示す．矩形描画に注目すると，SW-Py では 1 秒間にわずか 4.34 個の矩形しか描画できていない．つまり，10,000 回の描画に 2,300 秒以上を要している．それに対し，HW-BRAM では 1 秒あたり 1,400 個以上の矩形描画ができており，10,000 回の描画も 6.83 秒で完了する．これは，337 倍の高速化が達成されたことを意味する．この結果により，Python 単体では実用困難であった描画処理が，ハードウェア化によって実用レベルに引き上げられたといえる．直線・円・文字の描画についても，それぞれ 32.3 倍，449 倍，5.92 倍の高速化を達成した．また，ダブルバッファリング機構を用いる場合（HW-DB）と比較すると，スループットの低下はおおむね 5% 程度にとどまった．一方，OpenCV を使った場合とは 5〜14 倍程度低いスループットとなった．しかしながら，60 fps の動画に対して 1 フレームあたり 20 個以上の図形を描画できることを考えると，提案システムは実用上十分な性能を有しているといえる．

5.5. レイテンシの評価結果

各図形におけるレイテンシの一覧を表 2 に示す．フレームバッファ転送を伴う HW-DB や SW-CV と比較して，HW-BRAM のレイテンシが突出して低いことが見て取れる．具体的には，描画する図形にかかわらず，HW-BRAM は 1 ms 以内で描画を完了している．あるいは，フレームバッファ転送には少なくとも 1 回あたり 10 ms 強を要するともいえる．したがって，センサ値などの即時表示が求められる用途において，ティアリングを考慮する必要がなければ，提案手法の HW-BRAM モードを使うことが望ましいと考えられる．これにより，既存のライブラリよりも極めて高い応答性を提供できる．

6. おわりに

本研究では，SoC 型 FPGA 環境における可視化の性能と導入コストのトレードオフを解消するため，PYNQ 上で動作する可視化システムの開発支援ツールキットを構築した．本ツールキットは，高位合成により実装された高速な図形描画 IP と，回路導入および制御を自動化・抽象化するソフトウェア群から構成される．

表 2　各描画対象における 10,000 回描画時のレイテンシ（単位: ms）

図形種別	HW-BRAM	HW-DB	SW-CV	SW-Py
直線 (Line)	0.58	11.44	10.87	30.13
矩形 (Rectangle)	0.68	11.55	10.88	251.22
円 (Circle)	0.65	11.43	10.87	295.68
文字 (Text)	0.32	11.20	10.89	12.78

回路構築自動化スクリプトの検証の結果，提案スクリプトは既存回路に対して安全かつ確実に可視化機能を導入できることが確認された．これにより，複雑な配線作業を省略した，手軽な可視化システムの導入を実現した．

性能評価の結果，提案手法は従来の Python ソフトウェア実装と比較して最大約 449 倍の高速化を達成し，実用的な描画性能を持つことを確認した．また，既存の OpenCV 実装と比較してスループットでは劣るものの，レイテンシにおいては約 0.32 ms という圧倒的な応答速度を実現した．

以上により，ハードウェア実装による高速な描画処理と，ソフトウェアライブラリによる高いユーザビリティの両立を，PYNQ 環境において実現した．

今後の課題として，アルファブレンド機能の追加による表現力の向上や，Python から FPGA へのレジスタアクセスにおけるオーバーヘッド削減が挙げられる．

文　献

[1] S. Sarkar, S.S. Bhairannawar, and Raja K.B., “FPGACam: A FPGA based efficient camera interfacing architecture for real time video processing,” IET Circuits, Devices & Systems, vol.15, no.8, pp.814–829, 2021.

[2] N. Fujieda and N. Ito, “A case for edge video processing with FPGA SoC: reversi board detection using Hough transform,” 2024 Twelfth International Symposium on Computing and Networking Workshops (CANDARW), pp.50–55, 2024.

[3] Advanced Micro Devices, “PYNQ — Python Productivity for AMD Adaptive Computing platforms. [Online],” Available: `https://www.pynq.io/`. Accessed: 2026-02-04.

[4] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, “Real-time computer vision with OpenCV,” Communications of the ACM, vol.55, no.6, pp.61–69, 2012.

[5] J.F. Hughes, A. vanDam, M. McGuire, D.F. Sklar, J.D. Foley, S.K. Feiner, and K. Akeley, Computer Graphics: Principles and Practice, 3rd edition, Addison-Wesley Professional, 2013.

[6] Advanced Micro Devices, “Vitis High-Level Synthesis User Guide, UG1399 (v2025.2),” 2025.

[7] 門真なむ, “8 × 8 ドット日本語フォント「美咲フォント」[Online],” Available: `https://littlelimit.net/misaki.htm`. Accessed: 2026-02-04.

[8] Advanced Micro Devices, “Vivado Design Suite Tcl Command Reference Guide, UG835 (v2025.2),” 2025.

[9] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, “Caffeine: Toward Uniformed Representation and Acceleration for Deep Convolutional Neural Networks,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.38, no.11, pp.2072–2085, 2019.