初学者向け論理回路設計 Web エディタ「CircuitFlux」の開発と性能評価

天野 雄太[1,a)]　藤枝 直輝[1,b)]

概要：近年，初等中等教育におけるプログラミング教育の必修化に伴い，コンピュータの動作原理を理解するための第 1 歩である，論理回路の学習の重要度も高まっている．しかし，既存の学習ツールには環境構築・言語・操作性という 3 つの障壁が存在し，初学者の学習を阻害している．本研究では，これらの課題を解決するため，Web ベースの初学者向け論理回路エディタ「CircuitFlux」を開発した．本エディタは，日本語 UI による直感的な操作に加え，視認性を最優先した独自の自動配線アルゴリズム，およびリアルタイムなシミュレーションと HDL 自動生成機能を備える．本稿では，提案エディタの設計と実装について述べたあと，利用者に対するユーザビリティ評価の前段階として，自動配線および解析処理の性能評価について報告する．

キーワード：論理回路設計，教育支援システム，Web アプリケーション，自動配線，HDL

Depelopment and Performance Evaluation of “CircuitFlux,” A Logic Circuit Editor for Beginners

Yuta Amano[1,a)]　Naoki Fujieda[1,b)]

Abstract: As programming education has become compulsory in elementary and secondary education, the importance of learning logic circuits has been increasing, which is the first step to understand operating principles of computers. However, existing learning tools have three barriers against beginners: environment setup, language, and usability. In this research, we developed “CircuitFlux,” a Web-based logic circuit editor designed for beginners. The editor is easily accessible via a Web browser. It features intuitive operation via a Japanese UI, its own auto-routing algorithm that prioritizes visual clarity, and functionality of real-time simulation and automatic HDL generation. In this paper, we describe the design and implementation of the proposed editor and report on a performance evaluation of the auto-routing and analysis processes, as preliminary steps toward usability evaluation.

Keywords: Logic Circuit Design, Educational Support System, Web Application, Auto-Routing, HDL

1. はじめに

近年，初等中等教育の段階からのプログラミング教育が重視されている．その目的は，単にプログラミング言語を学ばせるというよりも，論理的思考力を育み，現代社会を支えるソフトウェア群の動作の仕組みを理解させることにある [1]．これに伴い，教育現場では Scratch などのビジュアルプログラミング言語や，Micro:bit などのフィジカルコンピューティング教材が広く活用されている．しかし，ソフトウェアの仕組みを真に理解するには，その土台であるコンピュータの動作原理の理解が不可欠である．現代のコンピュータが行う複雑な処理も，その構成要素を分解すれば NAND に代表される単純な論理ゲートの集合体へと帰着する [2]．つまり，論理回路の学習は，コンピュータの理解のための第 1 歩である．

[1] 愛知工業大学
Aichi Institute of Technology
[a)] v24702vv@aitech.ac.jp
[b)] nfujieda@aitech.ac.jp

© 2026 Information Processing Society of Japan

表 1: 既存ツールと本研究の機能・特徴比較

カテゴリ	環境構築 (Web)	言語 (日本語)	操作性 (自動配線)	視認性重視の配線
デスクトップ・教育用	×	△	×	-
Web・教育用	◯	×	△	×
Web・コード中心	◯	×	-	-
プロ向け	×	×	◯	×
本研究 (CircuitFlux)	◯	◯	◯	◯

論理回路の学習には，回路図の設計とシミュレーションにより動作を学習者に提示できるツールが有効である．しかし，標準的に利用されている既存の学習ツールには，特に初学者が利用する上で大きな障壁が存在する．本研究では，これらの障壁を環境構築・言語・操作性の 3 点にまとめる．環境構築において，一部のツールは特定の OS に依存しており，インストールや環境設定が学習開始の妨げとなる．Web ベースの手軽なツールも存在するが，その多くは日本語に対応しておらず，日本語話者にとっては言語の問題が操作習得の障害となりうる．手動配線の煩雑さや，視認性を考慮しない自動配線は操作性を損ない，初学者が本質的な回路設計へ集中することを阻害している．

そこで本研究では，これらの障壁を解決するため，Web ブラウザ上で動作する直感的な論理回路エディタ **CircuitFlux** を提案する．本エディタは，JavaScript で開発され，OS に依存せずインストール不要で動作することで，環境構築の障壁を取り除く．また，ユーザインタフェースの完全な日本語化により言語の障壁を解消する．さらに，操作性の障壁を解決するため，視認性を最優先した独自の自動配線アルゴリズムを実装し，直感的かつ効率的な学習環境を提供する．

本稿の構成は以下の通りである．2 節では関連研究について述べ，既存ツールの課題と本研究の位置づけを明確にする．3 節では，提案エディタ CircuitFlux の概要と主要機能について述べる．4 節では，本エディタの核となる自動配線アルゴリズムおよびリアルタイムシミュレーションの実装詳細について解説する．5 節でエディタのパフォーマンス評価を行い，6 節で考察する．7 節でまとめと今後の展望を述べる．

2. 関連研究

2.1 既存ツールの分類と課題

論理回路設計を支援する既存のツールは，その動作プラットフォームや学習目的によって大きく 4 つに分類できる．

第 1 に，デスクトップ・教育用ツールである．Deeds [3] や Logisim [4] などが代表的であり，教育的な回路設計に適している．しかし，これらは特定の OS へのインストールが必要であり，環境構築が初学者の障壁となる．また，配線操作が手動であることが多く，大規模な回路では操作が煩雑になる課題がある．

第 2 に，Web ベース・教育用ツールである．CircuitVerse [5] がその一例である．ブラウザ上で動作するため環境構築は不要だが，UI が英語のみである場合が多く，言語の障壁が存在する．また，自動配線機能を持たないか，持っていたとしても簡易的な L 字配線に留まり，回路図の視認性が低い場合がある．

第 3 に，Web ベース・HDL 学習用ツールである．EDA Playground [6] などが挙げられるが，これらはハードウェア記述言語（HDL）を用いたコーディングを中心としており，初学者が回路図から直感的に学ぶトップダウン型のアプローチには適していない．

第 4 に，汎用的なデスクトップ・電子回路設計（EDA）ツールである．本来は論理回路の学習を主目的としたものではないが，配線技術の高度な先行例として KiCad [7] 等が挙げられる．これらのツールは強力な自動配線機能を持つが，その主目的はプリント基板（PCB）の製造効率や信号特性の最適化にある．そのため，生成される配線は高密度で複雑になりがちであり，教育用回路図に求められる視認性，つまり見た目の分かりやすさを重視する本研究とは，設計思想が根本的に異なる．

2.2 本研究の位置づけ

本研究で提案する CircuitFlux は，これらの課題をすべて解決する独自の立ち位置にある．表 1 に，既存ツールと本エディタの比較を示す．本エディタは Web ベースかつ日本語 UI により環境と言語の障壁を取り除く．さらに，視認性を最優先した独自の自動配線アルゴリズムを実装することで，操作性の障壁を解消する点が最大の特徴である．

2.3 自動配線アルゴリズムの選定

本研究における自動配線は，視認性の最大化を目的とした，2 次元グリッド上での経路探索問題と位置づけられる．ここで視認性が高いとは，障害物との適切な距離を保ちつつ，方向転換の回数がより少なく，直線性が維持されていることと定義する．静的なグリッド空間における探索手法として，ダイクストラ法 [8] や A*アルゴリズム [9] などが知られている．本研究では，ヒューリスティック関数によりゴール方向への探索を効率的に導ける点，およびコスト関数の設計により曲がり角へのペナルティを柔軟に設定で

© 2026 Information Processing Society of Japan

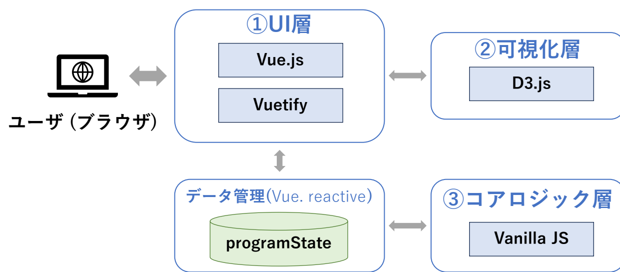


図 1: 提案エディタの構成図

きる点から，A*アルゴリズムを採用する．さらに，単純なA*探索では探索空間が広大になるため，第 1 フェーズとして矩形領域分割による大域的な経路探索を行う，ハイブリッド手法を提案する．詳細は 4 節で述べる．

3. 提案エディタの概要

本研究では，前述した 3 つの障壁（環境・言語・操作性）を解決するため，Web ブラウザ上で動作する論理回路エディタ CircuitFlux を開発する．提案エディタは, HTML5, CSS3, JavaScript (ES6+) 等の標準的な Web 技術を用いて構築されている．

3.1 エディタ構成

本エディタは，単一のページで動作する SPA (Single Page Application) として実装されている．提案エディタの構成を図 1 に示す．

UI 層では，フレームワークとして Vue.js [10]，UI コンポーネントライブラリとして Vuetify [11] を採用し，応答性の高いマテリアルデザインを実現している．可視化層におけるシミュレーション結果のタイムチャート表示には，データ可視化ライブラリである D3.js [12] を利用し，SVG ベースの動的な波形描画を行っている．コアロジック層，すなわち回路の描画 (SVG)・解析・シミュレーション・HDL 生成などの核心部分は，特定のフレームワークに依存しない Vanilla JS で実装し，処理の最適化と高速化を図っている．

これら各層の仲介を行うデータ管理部には，Vue.js のリアクティブシステムを採用している．図 1 に示す通り，UI 層でのユーザ操作やコアロジック層でのシミュレーション結果は，すべてこのデータ管理部へと集約される．これを各層へ即座に伝播・同期させることで，回路の編集から波形表示までの一連の処理をリアルタイムに実行できる構成としている．

3.2 データ構造

本エディタのデータ構造は，回路全体の状態を単一のグローバルオブジェクト (図 1 の programState) で管理する設計となっている．最大の特徴は，回路素子と配線が，互いの接続情報を ID で参照し合う双方向リンク構造にある．素子は自身のポートに接続された配線の ID を保持し，配線は自身に接続された素子の ID を保持する．この構造により，シミュレーション時の信号伝播や，HDL 生成時のネットリスト構築を $O(1)$ オーダーの参照で高速に行う．

3.3 主な機能

本エディタが提供する主要な学習支援機能を以下に示す．なお，自動配線，シミュレーション，HDL 生成の実装詳細については 4 節で述べる．

（1） **回路図エディタ機能**
SVG を用いたキャンバス上に，論理ゲートや入出力素子を配置し，マウス操作で回路を設計できる．

（2） **自動配線機能**
始点と終点をクリックするだけで，障害物を回避した視認性の高い経路を自動生成する．

（3） **リアルタイムシミュレーション**
回路の変更に合わせて即座に信号値を計算し，配線の色やインジケータで可視化する．3 値論理 (‘0’, ‘1’, ‘X’) に対応している．

（4） **HDL・双方向ハイライト**
回路図と等価な HDL コード (VHDL/Verilog) をリアルタイムに生成する．回路図上の素子とコード上の記述を相互にハイライトし，対応関係を明示する．

（5） **GUI ベースのテストベンチ生成**
GUI ベースでテストケースを設定し，回路に対応するテストベンチを生成する．生成したテストケースでシミュレーションする機能も実装している．

（6） **タイムチャート表示**
シミュレーション結果を時系列の波形グラフとして表示し，順序回路の動作確認を支援する．

4. 実装

本エディタは，Web 標準技術である HTML5，CSS3，および JavaScript (ECMAScript 2015 以降) を用いて開発した．本章では，主要機能の実装詳細について述べる．

4.1 UI/UX の実装

機能（1）の回路図エディタ機能は，初学者が迷わず直感的に操作できることを最優先に設計されている．提案エディタの読み込み完了直後の画面の例を，図 2 に示す．画面構成はヘッダ，左サイドバー，キャンバス，右サイドバー，ボトムバーからなる．左サイドバーは配置可能な素子が並べられた素子パレットであり，キャンバス上にこれらを配置・配線して作図する．右サイドバーには対応する

© 2026 Information Processing Society of Japan

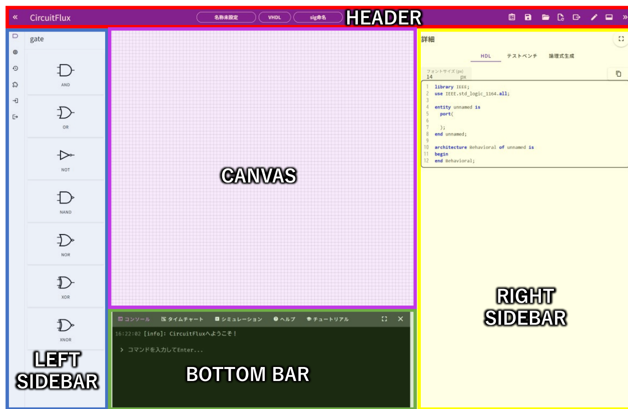


図 2: 起動直後のエディタ画面

HDL コードやプロパティが表示され，ボトムバーにはシミュレーションの結果やヘルプなどが表示される．

回路図の描画には SVG (Scalable Vector Graphics) を採用する．これにより，拡大・縮小しても描画が劣化せず，常に高い視認性を維持できる．また，Vue.js のリアクティブシステムにより，配線モードへの切り替えや素子のドラッグ移動といったユーザ操作に対し，DOM を効率的に更新して滑らかなインタラクションを実現している．

4.2 自動配線アルゴリズムの実装

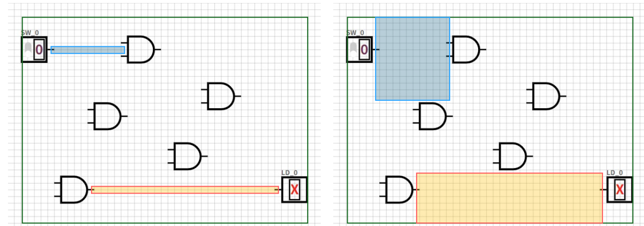
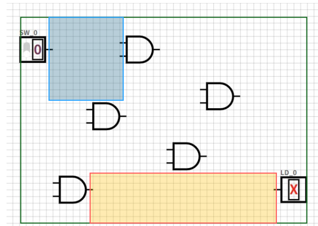
本研究では，配線の始点と終点を指定するだけで，障害物を回避しつつ視覚的に自然な経路を生成する自動配線アルゴリズムを提案し，提案ツールの機能（2）として実装する．高速性と経路の見た目の自然さを両立するため，以下の 2 段階のハイブリッドアプローチを採用する．以下の説明では，図 3 を用いて探索のプロセスを例示する．

4.2.1 第 1 フェーズ：矩形領域分割による大域探索

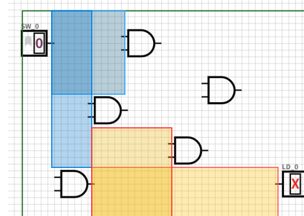
第 1 フェーズでは，障害物のない大まかな通路を高速に発見することを目的とする．グリッド単位で探索する従来の A*法に対し，本手法では障害物のない矩形領域を単位として探索することで，計算量を削減する．また，始点と終点の両側から同時に探索を進める双方向探索を導入することで，空間をさらに圧縮し，処理の高速化を図る．

第 1 フェーズではまず，始点と終点の 2 点を初期の探索点ペアとして，矩形の探索を開始する．各ステップではまず，初期の探索点ペアによって作られる長方形の長手方向に向けて，障害物にぶつかるまで辺を伸ばす．図 3 (a) では始点・終点から水平方向に辺を伸ばしている．次に，この辺を短手方向に障害物にぶつかるまで拡張し，通行可能な最大の矩形を生成する．図 3 (b) では，始点・終点からそれぞれ青と赤で示す矩形が生成される．

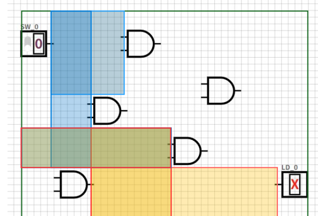
次に，探索の効率化と障害物回避のため，方向に優先度をつけた探索により新たな探索点を決定する．図 3 (a)(b) に示したように，直近の矩形拡張が水平方向であったとして説明する．このとき，生成された矩形の水平方向の辺

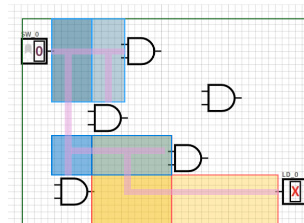
(a) 矩形の拡張（横）　(b) 矩形の拡張（縦）



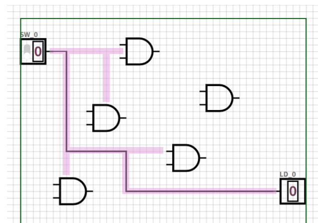
(c) 探索の続行



(d) 矩形の接触



(e) A*アルゴリズムによる経路探索



(f) 導き出された経路

図 3: 探索例の可視化

(上辺および下辺) において，垂直方向に障害物が隣接しない地点を走査する．その中から，最も目標地点に近い座標を次ステップの探索起点とする．もし水平方向の辺に有効な地点が見つからない場合は，矩形の垂直方向の辺 (右辺および左辺) を走査し，通行可能な代替地点を新たな探索起点とする．なお，直近の矩形拡張が垂直方向であった場合は，これまでの説明の水平を垂直，垂直を水平と，それぞれ読み替える．このように，原則として「水平→垂直→水平」と探索方向を交互に切り替えるジグザグの探索を優先しつつ，障害物がある場合のみ同一方向の拡張を継続する．これにより，広大なグリッド空間を最小限の矩形数で効率的に踏破する設計となっている．

始点側と終点側からの矩形群が接触した時点で，経路成立とみなし，探索を終了する．図 3 では，(c) で 2 回目の矩形を拡張し，(d) で 3 回目の矩形拡張で矩形同士の接触により，探索を終了している．接触した一連の矩形群において，矩形同士が重なり合う領域を「交差点」として抽出し，第 2 フェーズへと渡す．この「交差点」内は，方向転換が他の箇所よりも容易に行える場所とみなせる．

4.2.2 第2フェーズ：A*アルゴリズムによる最適経路決定

第2フェーズでは，第1フェーズで特定された「交差点」情報を利用し，A*アルゴリズムを用いて詳細な経路を決定する (図3 (e))．ここでは，最短経路ではなく視認性の高い経路を生成するために，独自のコスト関数を導入している．

導入したコスト関数では，通常の移動コストに加え，方向転換に対してペナルティを与える．特に，第1フェーズで抽出した「交差点」以外での方向転換には極めて高いペナルティを課す．これにより，細い通路では直進し，曲がるときは広い場所で曲がるという，人間が描くような自然な経路が選択される．得られた経路が図3 (f) である．人間が描くような自然な経路が導出されている．

また，経路決定を高速に行うために，探索ノードの管理（OPEN リスト）には最小ヒープを，探索済みノードの管理（CLOSE リスト）にはハッシュマップを使用している．これにより，ノードの取り出しと参照をそれぞれ $O(\log V)$ および $O(1)$ で行い，JavaScript 上でも高速な動作を実現している．

4.3 回路解析の実装

機能（3）のシミュレーションや機能（4）の HDL 生成では，共通の前処理として回路全体の接続依存関係の解析が行われる．本エディタでは，回路規模が増大しても高速に動作するよう，以下のアルゴリズムを採用した．

まず幅優先探索 (BFS) を用いて，入力から出力への依存関係に基づく処理順序（トポロジカルソート）を決定する．大多数の正常な回路ではこの処理のみで解析が完了するため，計算量は $O(V+E)$（V: 素子数，E: 配線数）となり極めて高速である．

BFS で順序決定ができなかった場合のみ，深さ優先探索 (DFS) を実行してループ箇所を特定する．ループ経路上に仮想的なフリップフロップを挿入することで，シミュレーションの発散を防ぎ，安定した動作を保証する．

4.4 シミュレーションエンジンの実装

機能（3）のシミュレーションでは，解析された処理順序に基づき，回路の動作を計算する．教育用途に適した挙動を実現するため，以下の特徴を持たせている．

シミュレーション結果の表現は，‘0’ (Low)，‘1’ (High) に加え，未接続や競合状態を示す ‘X’ (不定) の3値で扱う3値論理シミュレーションを採用した．これにより，初期化忘れや配線ミスなどを視覚的にフィードバックできる．信号値は JavaScript のプリミティブ値を用いて管理し，’0’ および ’1’ は数値の 0 および 1 に対応させ，不定値 ’X’ は値が存在しないことを示す null にマッピングする．各論理ゲートの演算は，静的メソッドを持つクラスとして実装する．例えば AND 演算の実装では，JavaScript の配列メソッド includes を活用している．まず入力配列に 0 が含まれる場合は即座に 0 を返し，次に null が含まれる場合は null を，それ以外（すべて 1）の場合のみ 1 を返すという優先順位に基づいたアルゴリズムを採用する．これにより，3値論理における「一つでも 0 ならば 0」「未知の値があれば結果も未知」という振る舞いを効率的に実現している．

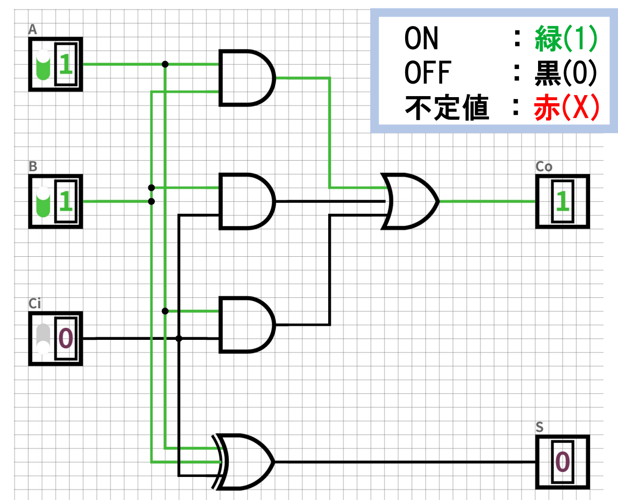


図 4: シミュレーション結果の可視化（全加算器）

シミュレーション実行順序は，回路解析で得られたトポロジカルソート済み配列を元に，順番にシミュレーションするレベル化シミュレーションを採用する．JavaScript 上でのシミュレーションは，前述の配列を forEach メソッドで走査する単一のループ処理として実装されている．ループ内では各素子オブジェクトの評価メソッドを呼び出し，3値論理演算クラスを用いて自身の出力値を確定させる．この評価は配列のインデックス順に行われるため，イベント駆動方式で見られる信号伝搬の不必要な再計算を回避し，最小限のステップ数で全出力値を確定できる設計となっている．

計算された信号値に応じて，配線の色（0: 黒，1: 緑，X: 赤）や LED の点灯状態を即座に更新し，結果の可視化を行っている（図4）．JavaScript からシミュレーション結果を，SVG 要素の stroke（色）属性を直接書き換えて色を変えている．

4.5 HDL 自動生成の実装

機能（4）の HDL 生成は，図とコードの対応を直感的に理解させるための工夫を凝らしている．

回路図からのコード生成は,「双方向リンク構造」を持つデータオブジェクトをスキャンすることで実現している．回路内の全ての素子と配線を走査し，それらの接続情報を言語ごとのテンプレートに当てはめていく手法をとる．これにより，素子が追加・削除されるたびに内部のジェネレータが走り，常に最新の HDL コードが表示エリアに反

© 2026 Information Processing Society of Japan

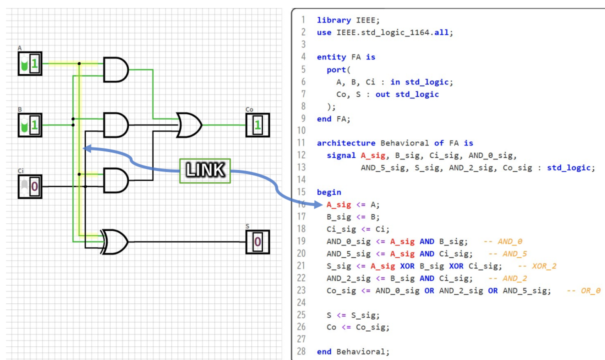


図 5: HDL の自動生成と双方向ハイライト

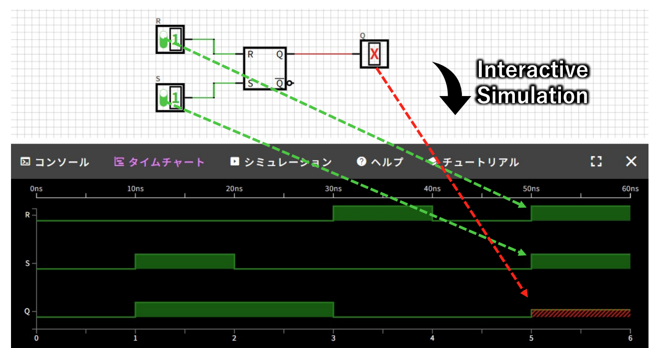


図 6: タイムチャート表示

映される仕組みとなっている．

図とコードの対応関係をより明確にするため，両者を相互にハイライトする機能も実装している．図 5 に，ワイヤーが黄色で，対応する HDL が赤色で，それぞれハイライトされる様子を示す．この機能を実現するため，コード生成時に各行の HTML 要素へ，対応する素子の ID を独自属性（data-id）として埋め込んでいる．回路図上の素子をクリックすると，その ID を持つ行をコードエリア内から検索し，ハイライト用の CSS を適用する．コードの特定の行をクリックすると，埋め込まれた ID を読み取り，キャンバス上の該当素子をハイライトする．

これらの機能をブラウザ上で高速に動かすため，内部的には表示用のプレーンテキストとハイライト・ID 埋め込み済みの HTML の 2 系統を同時に組み立てる処理を行っている．また，フリップフロップのような複雑な構成を持つ部品については，クロックやリセットの有無に応じてコードの骨組みを動的に組み立てるジェネレータ関数を構築している．これにより，多様な回路構成に対しても，無駄がなく正確な HDL コードをリアルタイムに生成することを可能にとしている．

4.6 タイムチャート表示機能の実装

機能（6）のタイムチャート表示では，シミュレーション実行時の信号状態を履歴として保持するため，観測対象となる各素子のインスタンス内部に時系列データの配列を定義している．シミュレーションが実行されるたびに，各素子の現ステップにおける信号値（0, 1, null）をこの配列へ push する．蓄積されたデータ群の可視化には，データ可視化ライブラリである D3.js を用いる．配列内の離散的なステップデータを SVG の座標系にマッピングし，D3.js の curveStepAfter 補間を用いて描画している．これにより，論理回路特有の垂直な立ち上がり・立ち下がり（矩形波）を正確に表現した．不定値については，SVG の pattern 要素を用いた赤色の斜線パターンで描画領域を塗りつぶす実装を行い，正常な信号値と明確に区別して表示する工夫を施した．

4.7 テストベンチ自動生成

機能（5）のテストベンチ生成は，HDL 記述に不慣れな初学者でも，設計した回路の動作検証を容易に行えるようにするためのものである．ユーザは GUI 上で入力信号のパターン（High/Low の期間）を設定するだけで，VHDL または Verilog HDL のテストベンチコードを自動生成できる．生成方式には，全入力パターンを網羅する総当たりモードと，雛形のみ生成するテンプレートモード，任意の信号変化を指定できるカスタムモードを備えている．生成されたテストベンチは内部で即座にシミュレーションされ，結果がタイムチャートに反映されるため，コードを書かずとも検証作業が完結する．

テストベンチの生成は，4.5 節で述べた HDL 生成と同様の双方向リンク構造を走査して回路のインスタンス化を行うが，特筆すべきはカスタムモードにおける入力パターンの生成手法である．ユーザが指定した値と保持サイクルのペアを解析し，各言語の遅延構文を用いたプロセス文を動的に組み立てる．

生成されたテストシナリオは，コードとして出力されると同時に，内部の配列へと展開される．この配列を 4.4 節のレベル化シミュレーションエンジンの入力として供給する．カスタムモードの場合，ユーザが設定したシミュレート長分のクロックサイクル数を，それ以外のモードの場合，シミュレーション時間を配列長と同じクロックサイクル数とすることで，外部ツールに依存しないブラウザ内での動作検証を実現している．この一連の処理により，GUI 設定の変更が即座にタイムチャートの波形更新へと同期する設計となっている．

5. 評価

本研究で開発したエディタのコア機能である，機能（2）の自動配線・回路解析，機能（3）のシミュレーション，および機能（4）の HDL 生成について，その実行速度と実用性を検証するため，定量的評価を行う．なお，自動配線アルゴリズムについては，得られた実行時間について考察するために，理論的計算量の検討も併せて行う．

© 2026 Information Processing Society of Japan

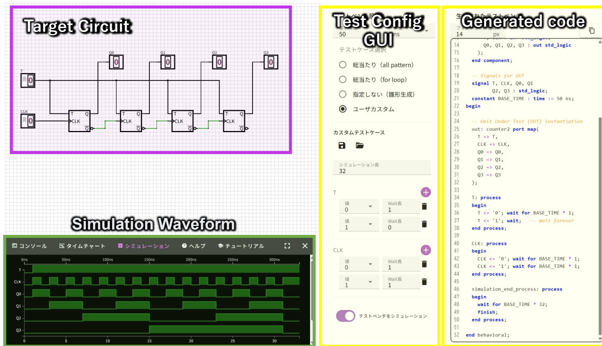


図 7: GUI 操作でのテストベンチ生成

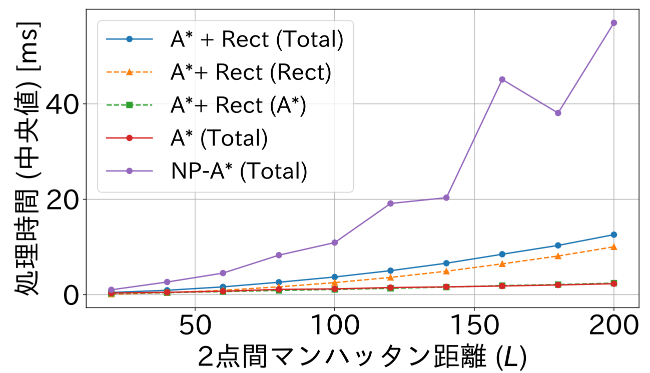


図 8: 配線距離と処理時間の関係（ケース 1）

5.1 理論的計算量の検討

本エディタの自動配線アルゴリズムについて，探索範囲の一辺のグリッド数を L とした際の理論的な計算量を評価する．

5.1.1 第 1 フェーズ（矩形領域分割）の計算量

本フェーズの計算量は，探索範囲の面積 L^2 と，生成される矩形の最大数 N に依存する．実装上，探索ループ回数には定数上限（30 回）を設けているため，N は定数とみなせる．各ステップにおける「巨大矩形の生成」は，探索方向および垂直方向への走査を行うため，最悪の場合でも $O(L^2)$ となる．したがって，第 1 フェーズ全体の計算量は $O(N \cdot L^2)$ となり，これは N が定数であることから $O(L^2)$ と評価できる．すなわち，探索範囲の面積に比例する計算時間で，大域的な経路（通路）を発見可能である．

5.1.2 第 2 フェーズ（A*探索）の計算量

第 2 フェーズにおける探索ノード数（グリッド数）を V $(= L^2)$ とする．A*アルゴリズムの効率は，OPEN リスト（未探索ノード）と CLOSE リスト（探索済みノード）のデータ構造に大きく依存する．本実装では，OPEN リストの管理に最小ヒープ を，CLOSE リストの管理にハッシュマップを採用した．これにより，ノードの取り出しは $O(\log V)$，探索済みチェックは平均 $O(1)$ で行えるため，全体の最悪計算量は $O(L^2 \log L)$ となる．

さらに，本研究のハイブリッドアプローチでは，第 1 フェーズで探索すべき通路が限定されるため，第 2 フェーズで実際に探索するノード数は全領域 L^2 よりも大幅に削減される．障害物が少ない典型的なケースにおいて，探索ノード数は経路長（$\approx L$）に比例する程度まで減少するため，実用上はより高速な動作が期待できる．

5.2 評価環境と計測手法

評価実験は以下の環境で実施した．

- **CPU**: 11th Gen Intel Core i5-11300H @ 3.10GHz
- **RAM**: 16.0 GB
- **Browser**: Google Chrome (Ver. 142.0)

JavaScript の実行時間は，JIT (Just-In-Time) コンパイラによる最適化の影響を大きく受ける．そこで，計測の安定性を確保するため，各テストケースの実行前に 30 回のウォームアップ走行を行い，その後の 300 回の実行時間の中央値を評価指標として採用した．なお，ブラウザのセキュリティ制限によるタイマー精度の低下を防ぐため，ローカルサーバーにて Cross-Origin Isolation を有効化し，高精度な計測環境を構築した．

5.3 自動配線アルゴリズムの評価

提案手法である矩形探索と A^* アルゴリズムの組み合わせ（以下，$A^* + \mathrm{Rect}$）の性能を評価するため，比較対象として単純な A^* アルゴリズム（以下，A^*）および屈曲へのペナルティを設けない A^* アルゴリズム（以下，NP-A^*）を用意し，以下の 3 つのケースで計測を行った．

(1) **ケース 1**: 障害物のない空間での長距離配線（距離 $L = 20 \sim 200$）

(2) **ケース 2**: 一般的な回路を模した，適度な障害物がある環境

(3) **ケース 3**: 障害物が密集した複雑な環境

5.3.1 実行時間の評価

ケース 1 における距離 L と実行時間の関係を図 8 に示す．A*は，障害物がない理想的な環境では最も高速であり，$O(L)$ であった．一方，A* + Rect は，第 1 フェーズ（矩形探索）のオーバーヘッドにより A*よりも時間を要するものの，距離 L に対して第 1 フェーズでの探索で $O(L^2)$，第 2 フェーズの探索で $O(L)$ を示している．5.1.2 項で述べた通り，第 1 フェーズによって探索すべき通路が限定されたことで，第 2 フェーズにおける実質的な探索ノード数が，全領域の $O(L^2)$ ではなく経路長に比例する $O(L)$ 程度まで削減されている．

表 2 に，ケース 2 およびケース 3 における計測結果を示す．障害物が存在する環境においても，A*が最も高速であった．これは，A*が独自のコスト関数によって探索空間を効率的に絞り込んでいるためである．一方で，A* +

© 2026 Information Processing Society of Japan

表 2: 自動配線アルゴリズムの計測結果（ケース 2・ケース 3）

テストパターン	Median Total [ms]	Median Rect [ms]	Median AStar [ms]	StdDev Total [ms]	CV Total	Turns
ケース 2（障害物あり）						
A* + Rect	6.332	2.540	2.740	0.622	0.096	4
A*	4.510	n/a	4.445	0.380	0.083	3
NP-A*	5.830	n/a	5.765	0.569	0.094	27
ケース 3（障害物あり (多)）						
A* + Rect	17.055	4.130	11.452	1.171	0.067	4
A*	14.685	n/a	14.617	1.235	0.083	3
NP-A*	8.540	n/a	8.472	1.536	0.171	19

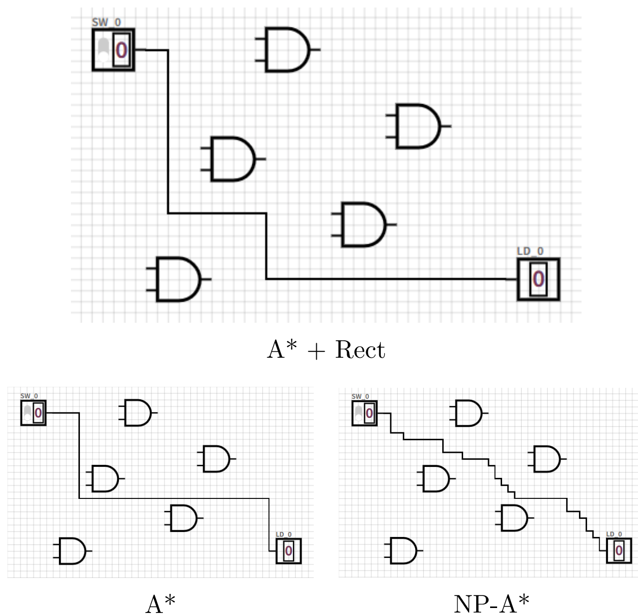


図 9: 生成された配線経路の比較（ケース 2）

Rect はケース 2 で 6.3 ms，最も複雑なケース 3 においても 17.1 ms という中央値を記録した．これは，Web ベースのエディタにおいてユーザが遅延を体感することのない十分なリアルタイム性能であると言える．

5.3.2　経路品質（視認性）の評価

生成された経路の比較を図 9 と図 10 に示す．NP-A*は，障害物を最短距離で回避しようとするあまり，不必要な屈曲が多発し，視認性が著しく低い．A*は，最短経路を選択する傾向があり，障害物に近接しすぎる場合がある．対して A* + Rect は，第 1 フェーズで抽出した広い通路を優先し，かつコスト関数により交差点以外での方向転換を抑制しているため，障害物から適度な距離を保ちつつ，曲がり角の少ない視認性の高い経路を生成できている．

5.3.3　回路解析・シミュレーション・HDL 生成の評価

回路規模の増大に対する各コア機能の応答性を評価するため，代表的な回路である N ビットカウンタおよび N ビット加算器を用いて，回路解析，シミュレーション，HDL 生成の各処理時間を計測した．それぞれの計測結果を図 11 および図 12 に示す．

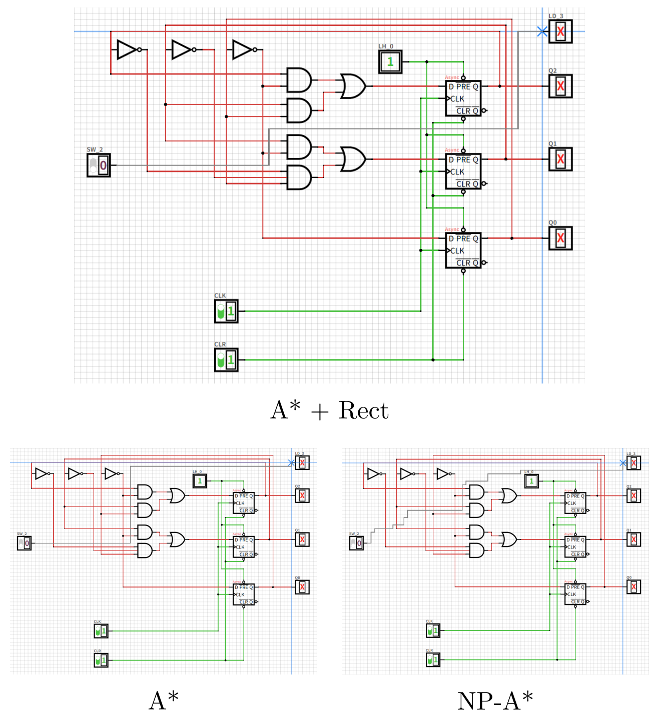


図 10: 生成された配線経路の比較（ケース 3）

回路解析処理はグラフより，素子数の増加に対して処理時間がほぼ線形に増加していることが分かる．これは，実装した BFS ベースのアルゴリズムが $O(V+E)$ で効率的に動作していることを示唆している．最大規模の 16 ビット加算器（130 素子）においても，解析は 4.8 ms 程度で完了しており，編集操作に対して瞬時に解析が終了することを確認した．

次に，入力信号の変化から全出力が確定するまでのシミュレーション時間を計測した．レベル化シミュレーションの採用により，こちらも素子数に対して線形の性能特性を示している．32 ビットカウンタの場合で約 1.7 ms，16 ビット加算器の場合で約 3.3 ms と，解析処理以上に高速であり，ユーザのインタラクティブな操作に対しても遅延なく追従可能である．

さらに，VHDL コード生成にかかる時間についても評価を行った．テキスト処理を含むため負荷が高いと予想され

© 2026 Information Processing Society of Japan

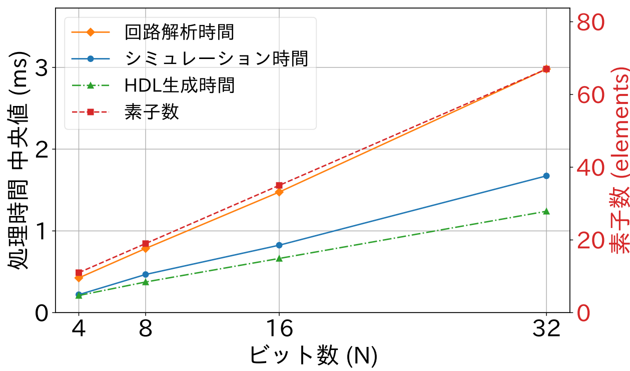


図 11: N ビットカウンタの処理時間プロット図

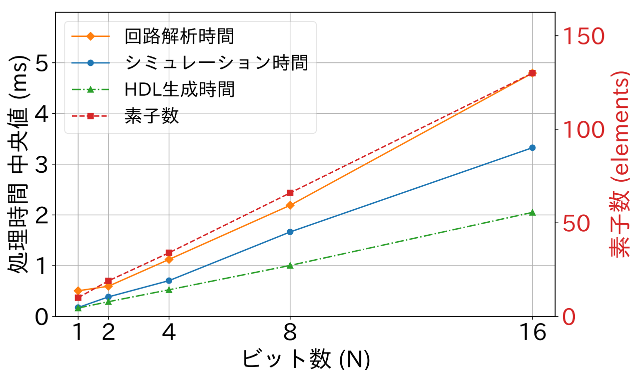


図 12: N ビット加算機の処理時間プロット図

たが，結果はシミュレーションと同等以上に高速であり，16 ビット加算器でも約 2.1 ms で生成が完了した．これにより，回路図を編集するたびにリアルタイムで HDL コードを更新・表示するという本エディタの仕様が，実用的な速度で実現できていることが実証された．

6. 考察

自動配線の評価結果（図 9，図 10）において，A* + Rect は A*よりも処理時間を要している．これは第 1 フェーズの矩形探索処理（$O(L^2)$）によるオーバーヘッドであるが，その代償として障害物からのマージン確保と不必要な屈曲の抑制という，視認性の向上を実現している．ケース 3 のような高密度な環境下でも中央値 17.1 ms という即時応答性を維持しており，学習支援ツールとしては，極限の速度よりも経路の品質（分かりやすさ）を優先した本手法の設計思想が妥当であると結論付けられる．

以上の結果より，本エディタのコアアルゴリズムは，学習用途で想定される回路規模（百素子程度）に対して十分なパフォーマンスを有していると言える．特に自動配線においては，計算速度を維持しつつ，教育的に重要な視認性を向上させることに成功しており，既存ツールに対する明確な優位性が確認された．また，解析・シミュレーション等の各処理が数ミリ秒オーダーで完了することから，Web ベースであってもネイティブアプリに遜色ない操作感を提供可能であることが示された．

7. おわりに

本研究では，論理回路学習における環境構築，言語，操作性の 3 つの障壁を解決するため，Web ブラウザ上で動作する初学者向けエディタ「CircuitFlux」を開発した．パフォーマンス評価の結果，自動配線を含む全てのコア機能が，学習で扱う規模の回路に対してリアルタイムに応答することが実証された．

今後は，本エディタの教育的効果を客観的に検証するため，学部 3 年生等の初学者を対象とした既存ツール（Deeds）との比較実験を行う予定である．具体的には，日本語 UI や環境構築不要の利便性が環境・言語の障壁をどの程度低減させたか，また自動配線や HDL ハイライト機能が操作性の障壁の解消と回路動作の理解にどう寄与したかを，アンケート調査を通じて定性・定量的に明らかにする．これらの検証で得られた知見やユーザからのフィードバックを開発へ反映させることで，操作性や学習支援機能のさらなるブラッシュアップを図り，より初学者のニーズに合致した学習環境の実現を目指す．

参考文献

[1] 文部科学省: 小学校プログラミング教育の手引（第三版），2020.
[2] Nisan, N. and Schocken, S.: コンピュータシステムの理論と実装 第 2 版 —モダンなコンピュータの作り方 (斎藤康毅 訳), オライリー・ジャパン, 2024.
[3] Donzellini, G. and Ponta, D.: From Gates to FPGA: Learning Digital Design with Deeds, *Proc. IEDEC 2013*, pp. 41–48 (2013).
[4] Burch, C.: Logisim: a graphical system for logic circuit design and simulation, *J. Educ. Resour. Comput.*, Vol. 2, No. 1, pp. 5–16 (2002).
[5] CircuitVerse: CircuitVerse - Online Digital Logic Circuit Simulator, `https://circuitverse.org/` (accessed 2024-12-26).
[6] Doulos: EDA Playground, `https://www.edaplayground.com/` (accessed 2025-10-24).
[7] The KiCad Development Team: KiCad (Version 9.0.5), `https://www.kicad.org` (accessed 2025-10-24).
[8] Dijkstra, E. W.: A note on two problems in connexion with graphs, *Numerische Mathematik*, Vol. 1, No. 1, pp. 269–271 (1959).
[9] Hart, P. E., Nilsson, N. J. and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. Syst. Sci. Cybern.*, Vol. SSC-4, No. 2, pp. 100–107 (1968).
[10] Vue.js Team: Vue.js, `https://vuejs.org/` (accessed 2025-10-25).
[11] Vuetify Team: Vuetify, `https://vuetifyjs.com/` (accessed 2025-10-25).
[12] Bostock, M., Ogievetsky, V. and Heer, J.: D^3: Data-Driven Documents, *IEEE Trans. Vis. Comput. Graph.*, Vol. 17, No. 12, pp. 2301–2309 (2011).
[13] Quine, W. V.: The Problem of Simplifying Truth Functions, *Am. Math. Mon.*, Vol. 59, No. 8, pp. 521–531 (1952).
[14] McCluskey, E. J.: Minimization of Boolean Functions, *Bell Syst. Tech. J.*, Vol. 35, No. 6, pp. 1417–1444 (1956).

© 2026 Information Processing Society of Japan