Blockpic Datasheet: Implemented Features

This document provides a hierarchical overview of the features implemented in the Blockpic DSL and rendering engine.

1. Core Language (DSL)

- **AST Structure**: Support for `Diagram`, `Block`, `Wire`, and `Layout` definitions.
- **Lexer/Parser**: Recursive descent parser capable of reading `.blockpic` files.
  - Block Attributes: `kind`, `inputs`, `outputs`, `params`, `size`.
  - Wire Attributes: `source`, `destination`, `thickness`, `crossline`, `hint`.
  - Layout Attributes: Explicit coordinate mapping (`at (x,y)`).

2. Rendering Engine

- **Base Graphics**: Integration with `draw2d` for vector rendering.
- **Aesthetics**: The default netlistsvg skin, `skins/skin_xilinx.svg`, follows the figures of AMD's UG474: Helvetica or Arial of regular weight, thin lines without arrowheads, pin names inside generic blocks and flip-flops (`D`, `CE`, `CK`, `Q`), small port flags, and plain multiplexer trapezoids. `skins/skin_with_arrows.svg`, the earlier bold monospace look with arrowheads, is chosen with the `skin` argument of `blockpic_diagram`.
- **Shapes Library**:
  - Basic Logic: `AND`, `OR`, `NOT` (IEEE standard geometry).
  - Sequential: `DFF` (D-Flip-Flop with clock tick).
  - Routing: `MUX` (Trapezoidal multiplexer).
  - Fallback: Standard rectangular block rendering.
- **Wire Routing**:
  - Support for `thickness` styling.
  - Bus-width marks: every wire wider than one bit is crossed by a short slash with its width beside it. The width is the wire's `crossline` when it has one, else its `thickness`. A wire with a `type: "name"` shows the type's name instead, and `blockpic import` takes it from the `wiretype` attribute that Yosys gives a wire declared with a SystemVerilog typedef.
  - Orthogonal routing paths (`straight`, `L-shape`, `Z-shape`).
  - Bus width indicators (`crossline`).

3. Layout Engine

- **Auto-Layout**: Basic flow-based coordinate assignment for unmapped blocks.

- **Groups**: ELK lays each `group` out as one box, so no other block falls inside its dashed outline. Groups that share a block cannot each be one box; their outlines are drawn around their blocks after the layout, as before.
- **Commutative inputs**: The inputs of an `AND`, `NAND`, `OR`, `NOR`, `XOR`, `XNOR`, `ADD`, `MUL`, `EQ` or `NE` block go to their wires in the order the wires arrive, the topmost input to the wire from highest, so that the wires do not cross on their way in. A change is kept only when the drawing has fewer crossings, or as many and fewer bends.
- **Straight single wires**: A block with one wire, such as a constant or a port, moves up or down until its wire runs straight to the pin at the other end, where it has room to.
- **No jogs**: A middle run of a wire that steps aside and back, as ELK leaves one that crosses a group's border, slides into line with the run before or after it, where it then runs through no block or label.

4. Tooling

- **CLI Application**: `blockpic` binary.
  - `blockpic compile <file>`: Validates DSL syntax.
  - `blockpic render <file> [-format png] [-o output]`: Generates diagram images.
  - `blockpic layout <netlist.json> -src <file.blockpic> -skin <skin.svg> -nsv <nsv> [-o layout.json] [-netlist arranged.json] [-v]`: Lays a rendered netlist out with ELK and improves the layout, as described in section 3, for `//tools/netlistsvg:nsv` to draw. It writes the layout to `-o`, and to `-netlist` the netlist to draw with it, whose commutative inputs may be reordered; `-v` says what it changed. `blockpic_diagram` runs it between `render` and `refine`.
  - `blockpic refine <file.blockpic> -svg <drawn.svg> [-layout layout.json] [-o out.svg]`: Finishes a drawing: thick buses with their width marks, the blocks' names, and the groups' boxes, where `-layout` puts them or else around their blocks.
  - `blockpic import <netlist.json> [-top MODULE] [-o out.blockpic]`: Writes a Yosys JSON netlist, such as the one `rules_openxc7`'s `vivado_synthesis` gives, as blockpic source. Each cell becomes a block of its type, and each module port an `INPUT_PIN` or `OUTPUT_PIN` block. A bus is one thick wire when all its bits go to one pin, and one pin per bit (`PORT_i`) when they do not. Constants become `GND`, `VCC` or `CONST` blocks, which the skins draw as tie-offs: the value beside a short stub at the pin.

5. Continuous Delivery

- **Automated Releases**: GitHub Workflow for multi-arch binary releases (Linux, macOS).

- **Versioning**: Automatic semantic version tagging.

6. IEEE Standard Shape Library

Blockpic provides a comprehensive library of digital logic blocks based on IEEE standards.

6.1 Supported Shapes

- **Logic Gates**: `AND`, `OR`, `NOT`, `XOR`.
- **Sequential Logic**: `DFF`, `JKFF`, `TFF`, `SRFF`.
- **Routing**: `MUX`, `DEMUX`.
- **Power/Signal**: `VCC`, `GND` (drawn as tie-offs), `CLOCK`.

6.2 Visual Reference and Source Code

```
diagram Shapes {
    block AND1 { kind: "AND"; inputs: A, B; outputs: O; }
    block OR1 { kind: "OR"; inputs: A, B; outputs: O; }
    block NOT1 { kind: "NOT"; inputs: I; outputs: O; }
    block XOR1 { kind: "XOR"; inputs: A, B; outputs: O; }
    block DFF1 { kind: "DFF"; inputs: D, CLK; outputs: Q; }
    block MUX1 { kind: "MUX"; inputs: I0, I1, S; outputs: O; }
    block DEMUX1 { kind: "DEMUX"; inputs: I, S; outputs: O0, O1; }
    block VCC1 { kind: "VCC"; outputs: V; }
    block GND1 { kind: "GND"; outputs: G; }
    block CLK1 { kind: "CLOCK"; outputs: C; }

    layout {
        AND1 at (20, 20)
        OR1 at (120, 20)
        NOT1 at (220, 20)
        XOR1 at (320, 20)
        DFF1 at (20, 100)
        MUX1 at (120, 100)
        DEMUX1 at (220, 100)
        VCC1 at (320, 100)
        GND1 at (20, 180)
        CLK1 at (120, 180)
    }
}
```

7. AST Validation

Blockpic includes a validation layer to ensure logical correctness before rendering. * **Unique Names**: Blocks must have unique identifiers. * **Connectivity**: Wire

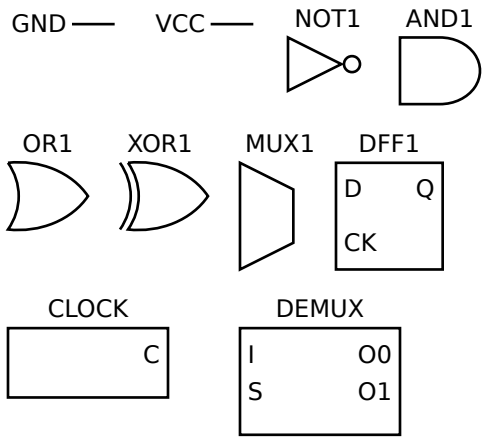

Figure 1: IEEE Shapes

source and destination blocks must exist. * **Pin Verification**: Referenced pins must exist on their respective blocks.

8. Pins and Buses

Blockpic supports explicit pin rendering and multi-bit bus visualization. * **Pin Rendering**: Named inputs and outputs are rendered on block boundaries. * **Bus Thickness**: Wires can have custom thickness to represent buses. * **Crossline Indicator**: Standard bus width indicators with labels.

8.1 Visual Reference and Source Code

```
diagram PinsAndBuses {
    block REG1 {
        kind: "REGISTER";
        inputs: D, CLK, EN;
        outputs: Q;
    }

    block MUX1 {
        kind: "MUX";
        inputs: I0, I1, S;
        outputs: O;
    }

    wire REG1.Q -> MUX1.I0 {
        thickness: 3;
```

```
        crossline: 32;
        hint: "Z-shape";
    }

    layout {
        REG1 at (20, 20)
        MUX1 at (150, 20)
    }
}
```

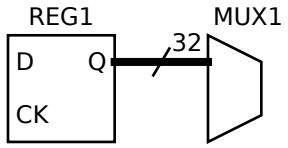

Figure 2: Pins and Buses

9. Dynamic Gate Sizing

Blockpic automatically calculates block dimensions based on the number of input and output pins. * **Automatic Height**: Blocks expand vertically to accommodate multiple pins while maintaining even distribution. * **Precise Pin Positioning**: Pins are positioned exactly at the boundary intersection, ensuring clean orthogonal routing.

9.1 Visual Reference and Source Code

```
diagram DynamicSizing {
    block LUT_MANY {
        kind: "LUT";
        inputs: I0, I1, I2, I3, I4, I5, I6, I7;
        outputs: O0, O1;
    }

    block SMALL {
        kind: "NOT";
        inputs: I;
        outputs: O;
    }

    wire LUT_MANY.O0 -> SMALL.I {
        hint: "Z-shape";
    }
```

```
    layout {
        LUT_MANY at (20, 20)
        SMALL at (200, 40)
    }
}
```

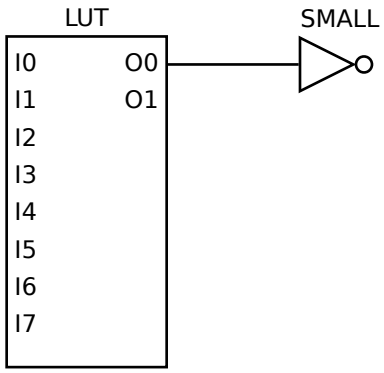

Figure 3: Dynamic Sizing

10. Registers, Counters, and Dependency Notation

Blockpic strictly adheres to IEEE Std 91-1984 for MSI components. * **Common Control Block**: MSI blocks like `REGISTER` and `COUNTER` feature a separated top section for common controls. * **Dependency Notation**: Pins are labeled using IEEE functional indicators (`C` for clock, `EN` for enable, `R` for reset, `S` for set). * **Dynamic Indicators**: Clocks (`CLK`, `C`) are rendered with the standard `>` wedge indicator.

10.1 Visual Reference and Source Code

```
diagram MSIBlocks {
    block REG8 {
        kind: "REGISTER";
        inputs: D [left], CLK [left], EN [left], R [left];
        outputs: Q [right];
        params: { width: "8" };
    }

    block CTR4 {
        kind: "COUNTER";
        inputs: CLK [left], EN [left], R [left];
        outputs: Q [right], TC [right];
        params: { width: "4" };
```

```
    }

    layout {
        REG8 at (20, 20)
        CTR4 at (200, 20)
    }
}
```

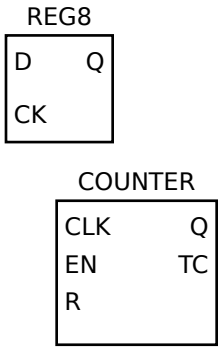

Figure 4: MSI Blocks

11. Pin-Aware Routing

Blockpic routes wires orthoganally, considering the physical side of the block where the pin is located. * **Directional Exits**: Wires exit perpendicular to the boundary they originate from. * **Directional Entries**: Wires enter perpendicular to the boundary they terminate at. * **Supported Hints**: `straight`, `L-shape`, `Z-shape` paths intelligently adapt to top/bottom/left/right pin orientations.

11.1 Visual Reference and Source Code

```
diagram PinAware {
    block SRC {
        kind: "CUSTOM";
        inputs: I_T [top], I_B [bottom], I_L [left];
        outputs: O_R [right], O_T [top], O_B [bottom];
        size: (60, 60);
    }

    block DST_TOP {
        kind: "CUSTOM";
        inputs: I_B [bottom];
    }
```

```
    block DST_BOT {
        kind: "CUSTOM";
        inputs: I_T [top];
    }

    block DST_RIGHT {
        kind: "CUSTOM";
        inputs: I_L [left];
    }

    wire SRC.O_T -> DST_TOP.I_B { hint: "L-shape" }
    wire SRC.O_B -> DST_BOT.I_T { hint: "L-shape" }
    wire SRC.O_R -> DST_RIGHT.I_L { hint: "straight" }

    layout {
        SRC at (100, 100)
        DST_TOP at (100, 20)
        DST_BOT at (100, 200)
        DST_RIGHT at (200, 100)
    }
}
```

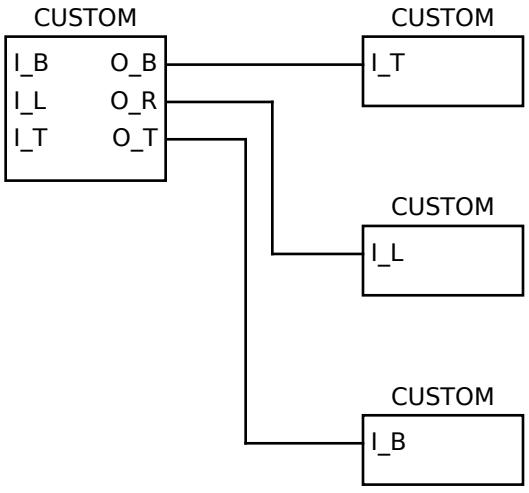

Figure 5: Pin-Aware Routing

12. Advanced Auto-Layout

Blockpic features a robust layer-based automatic layout engine. * **Dependency Grouping**: Blocks are grouped into columns (layers) based on their logical depth from inputs to outputs. * **Topological Sorting**: Longest-path layer assignment ensures clear left-to-right flow without manual coordinate mapping. * **Cycle Handling**: The algorithm gracefully handles feedback loops by assigning arbitrary starting layers to cyclic components.

12.1 Visual Reference and Source Code

```
diagram ComplexLayout {
    block IN1 { kind: "INPUT_PIN"; outputs: O; }
    block IN2 { kind: "INPUT_PIN"; outputs: O; }

    block AND1 { kind: "AND"; inputs: I1, I2; outputs: O; }
    block OR1 { kind: "OR"; inputs: I1, I2; outputs: O; }
    block NOT1 { kind: "NOT"; inputs: I; outputs: O; }

    block OUT1 { kind: "OUTPUT_PIN"; inputs: I; }

    wire IN1.O -> AND1.I1
    wire IN2.O -> AND1.I2
    wire IN2.O -> OR1.I1
    wire AND1.O -> OR1.I2
    wire OR1.O -> NOT1.I
    wire NOT1.O -> OUT1.I

    // No layout block provided, forcing AutoLayout to run
}
```

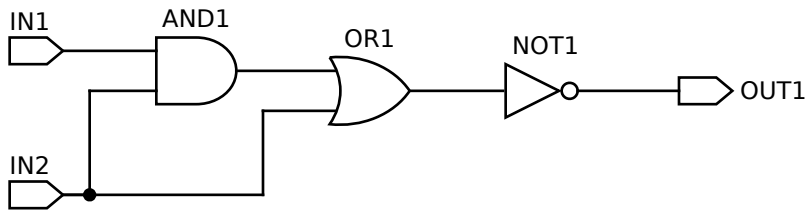

Figure 6: Complex Auto Layout

13. Netlistsvg Skins Integration

Blockpic supports mapping its blocks to the comprehensive set of standard `netlistsvg` skin components natively. This enables rich schematic diagrams

containing arithmetic operators, bitwise logic, latches, multiplexers, and specialized multi-input gates.

13.1 Arithmetic and Logic Operators

```
diagram skin_arithmetic {
  block IN_A { kind: "INPUT_PIN"; outputs: O; }
  block IN_B { kind: "INPUT_PIN"; outputs: O; }
  block OUT_Y { kind: "OUTPUT_PIN"; inputs: I; }

  block ADD { kind: "add"; inputs: A, B; outputs: Y; }
  wire IN_A.O -> ADD.A
  wire IN_B.O -> ADD.B
  wire ADD.Y -> OUT_Y.I

  block SUB { kind: "sub"; inputs: A, B; outputs: Y; }
  wire IN_A.O -> SUB.A
  wire IN_B.O -> SUB.B

  block MUL { kind: "mul"; inputs: A, B; outputs: Y; }
  wire IN_A.O -> MUL.A
  wire IN_B.O -> MUL.B

  block DIV { kind: "div"; inputs: A, B; outputs: Y; }
  wire IN_A.O -> DIV.A
  wire IN_B.O -> DIV.B

  block LT { kind: "lt"; inputs: A, B; outputs: Y; }
  wire IN_A.O -> LT.A
  wire IN_B.O -> LT.B

  block SHR { kind: "shr"; inputs: A, B; outputs: Y; }
  wire IN_A.O -> SHR.A
  wire IN_B.O -> SHR.B
}
```

13.2 Bitwise and Reduction Operators

```
diagram skin_bitwise {
  block IN_A { kind: "INPUT_PIN"; outputs: O; }

  block RED_AND { kind: "reduce_and"; inputs: A; outputs: Y; }
  wire IN_A.O -> RED_AND.A

  block RED_OR { kind: "reduce_or"; inputs: A; outputs: Y; }
  wire IN_A.O -> RED_OR.A
```

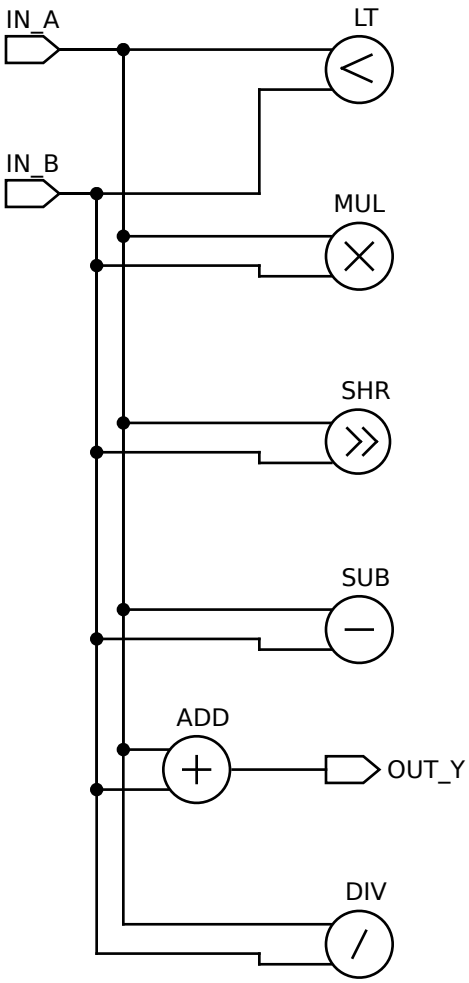

Figure 7: Arithmetic Components

```
  block RED_XOR { kind: "reduce_xor"; inputs: A; outputs: Y; }
  wire IN_A.O -> RED_XOR.A

  block LOGIC_NOT { kind: "logic_not"; inputs: A; outputs: Y; }
  wire IN_A.O -> LOGIC_NOT.A
}
```

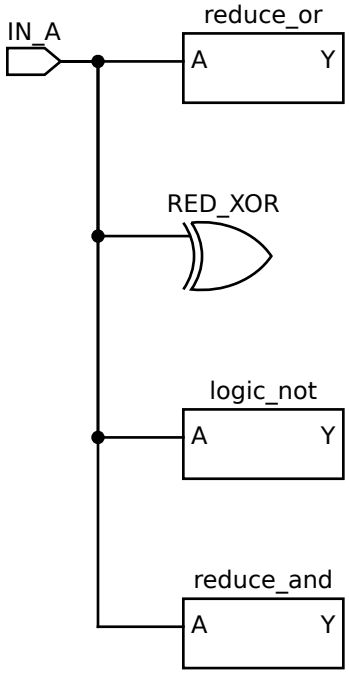

Figure 8: Bitwise Components

13.3 Latches and Registers

```
diagram skin_latches {
  block IN_D { kind: "INPUT_PIN"; outputs: O; }
  block IN_CLK { kind: "INPUT_PIN"; outputs: O; }

  block DFF { kind: "dff"; inputs: D, CLK; outputs: Q; }
  wire IN_D.O -> DFF.D
  wire IN_CLK.O -> DFF.CLK

  block DFF_BUS { kind: "dff_bus"; inputs: D, CLK; outputs: Q; }
  wire IN_D.O -> DFF_BUS.D
```

```
  wire IN_CLK.O -> DFF_BUS.CLK

  block DLATCH { kind: "dlatch"; inputs: D, EN; outputs: Q; }
  wire IN_D.O -> DLATCH.D
  wire IN_CLK.O -> DLATCH.EN
}
```

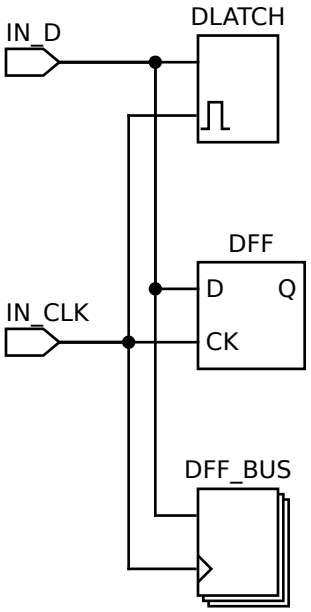

Figure 9: Latches and Registers

13.4 Multiplexers

```
diagram skin_muxes {
  block IN_A { kind: "INPUT_PIN"; outputs: O; }
  block IN_B { kind: "INPUT_PIN"; outputs: O; }
  block IN_S { kind: "INPUT_PIN"; outputs: O; }

  block MUX { kind: "mux"; inputs: A, B, S; outputs: Y; }
  wire IN_A.O -> MUX.A
  wire IN_B.O -> MUX.B
  wire IN_S.O -> MUX.S

  block MUX_BUS { kind: "mux_bus"; inputs: A, B, S; outputs: Y; }
  wire IN_A.O -> MUX_BUS.A
  wire IN_B.O -> MUX_BUS.B
```

```
  wire IN_S.O -> MUX_BUS.S
}
```

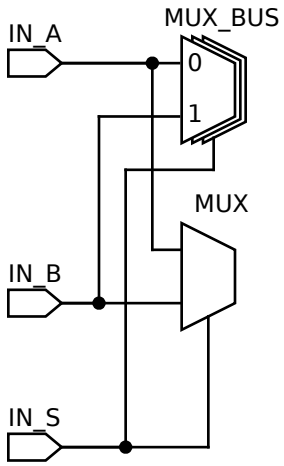

Figure 10: Multiplexers

13.5 Special Logic Gates

```
diagram skin_special {
  block IN_A { kind: "INPUT_PIN"; outputs: O; }
  block IN_B { kind: "INPUT_PIN"; outputs: O; }
  block IN_C { kind: "INPUT_PIN"; outputs: O; }
  block IN_D { kind: "INPUT_PIN"; outputs: O; }

  block AOI3 { kind: "aoi3"; inputs: A, B, C; outputs: Y; }
  wire IN_A.O -> AOI3.A
  wire IN_B.O -> AOI3.B
  wire IN_C.O -> AOI3.C

  block AOI4 { kind: "aoi4"; inputs: A, B, C, D; outputs: Y; }
  wire IN_A.O -> AOI4.A
  wire IN_B.O -> AOI4.B
  wire IN_C.O -> AOI4.C
  wire IN_D.O -> AOI4.D
}
```

13.6 Utility Nodes

```
diagram skin_utility {
  block IN_A { kind: "INPUT_PIN"; outputs: O; }
```

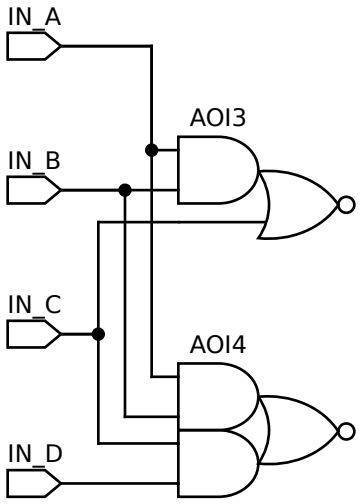

Figure 11: Special Logic Gates

```
  block IN_EN { kind: "INPUT_PIN"; outputs: O; }

  block TRIBUF { kind: "tribuf"; inputs: A, EN; outputs: Y; }
  wire IN_A.O -> TRIBUF.A
  wire IN_EN.O -> TRIBUF.EN

  block BUF { kind: "buf"; inputs: A; outputs: Y; }
  wire IN_A.O -> BUF.A

  block POS { kind: "pos"; inputs: A; outputs: Y; }
  wire IN_A.O -> POS.A
}
```

14. Advanced Routing and Text Features

Blockpic natively supports advanced ElkJS layout features via the integrated `netlistsvg` engine, including automatic cycle resolution for feedback loops. Additionally, text decorators such as unicode overbars (for active-low signals) and multiline parameters are fully supported. Furthermore, all wires are styled with directional arrowheads to clearly denote signal flow.

Source Code

Rendered Diagram

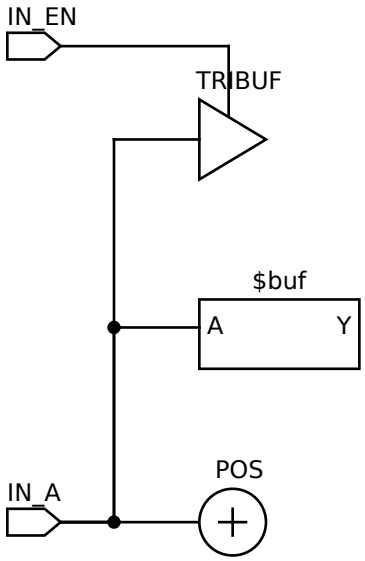

Figure 12: Utility Nodes

```
diagram TestFeatures {
    block IN_CLK { kind: "INPUT_PIN"; outputs: O; }
    block DFF1 {
        kind: "dff";
        inputs: D, CLK;
        outputs: Q;
        params: { "INIT": "0x00\nMODE: FAST", "LABEL": "~EN~" };
    }
    block NOT1 { kind: "not"; inputs: A; outputs: Y; }

    wire IN_CLK.O -> DFF1.CLK
    wire DFF1.Q -> NOT1.A
    wire NOT1.Y -> DFF1.D { hint: "feedback" }
}

diagram StateMachineFeedback {
    block CLK { kind: "INPUT_PIN"; outputs: O; }
    block RST { kind: "INPUT_PIN"; outputs: O; }
    block IN_A { kind: "INPUT_PIN"; outputs: O; }
    block OUT_Y { kind: "OUTPUT_PIN"; inputs: I; }

    block STATE_REG {
        kind: "dff";
```

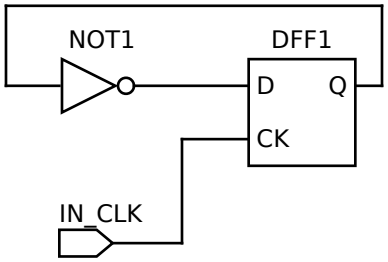

Figure 13: Test Features

```
        inputs: D, CLK;
        outputs: Q;
        params: { "LABEL": "STATE" };
    }

    block NEXT_STATE_LOGIC {
        kind: "CUSTOM";
        inputs: A, STATE_IN;
        outputs: NEXT_STATE;
        params: { "LABEL": "Next State\nLogic" };
        size: (80, 80);
    }

    block OUTPUT_LOGIC {
        kind: "CUSTOM";
        inputs: STATE_IN;
        outputs: Y;
        params: { "LABEL": "Output\nLogic" };
        size: (60, 60);
    }

    wire IN_A.O -> NEXT_STATE_LOGIC.A
    wire STATE_REG.Q -> NEXT_STATE_LOGIC.STATE_IN { hint: "feedback" }

    wire NEXT_STATE_LOGIC.NEXT_STATE -> STATE_REG.D
    wire CLK.O -> STATE_REG.CLK

    wire STATE_REG.Q -> OUTPUT_LOGIC.STATE_IN
    wire OUTPUT_LOGIC.Y -> OUT_Y.I
}
```

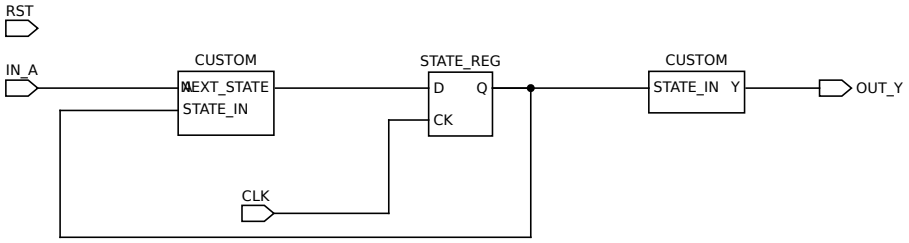

Figure 14: State Machine Feedback

15. Xilinx Layout Comparison

Blockpic supports high-fidelity schematic replication of common FPGA vendor diagrams. Below is a comparison between the original Xilinx UG474 `figure_1_1` documentation image and the exact identical layout recreated directly from Blockpic source code.

This demonstrates the engine's capability to render complex hierarchical layouts, custom element sizing, dashed group bounding boxes, and thick hollow block-arrow buses.

Original Xilinx Reference

Blockpic Rendered SVG

Blockpic Source Code

```
diagram figure_1_1 {
    block SwitchMatrix {
        kind: "generic"
        name: "Switch Matrix"
        inputs: in
        outputs: out0, out1
        size: (50, 100)
    }

    block Slice0 {
        kind: "generic"
        name: "Slice(0)"
        inputs: in, cin
        outputs: out, cout
        size: (50, 40)
    }

    block Slice1 {
        kind: "generic"
```

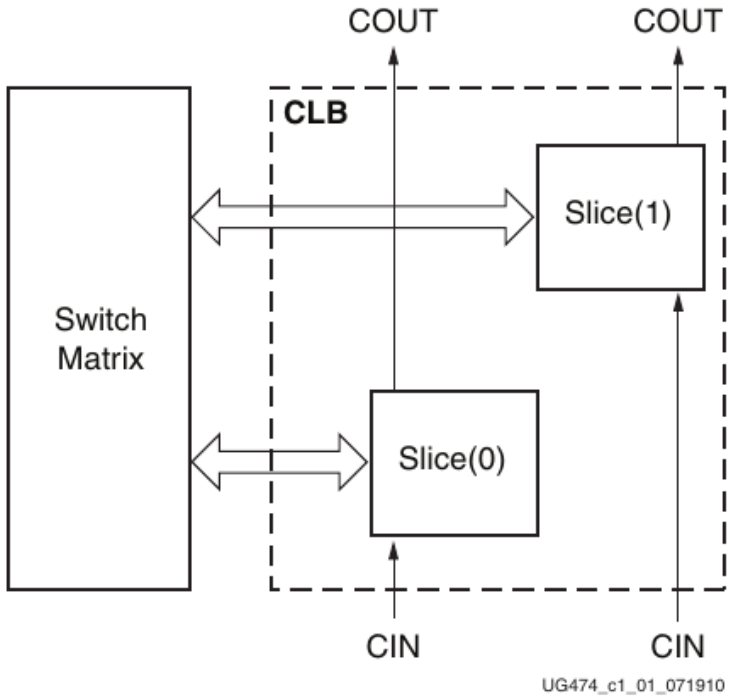

Figure 15: Original Xilinx UG474 Figure 1.1

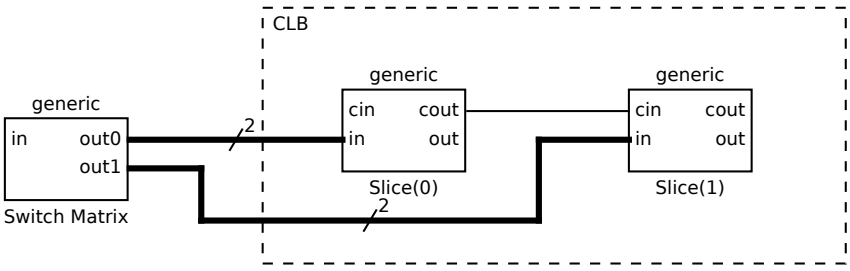

Figure 16: Blockpic Figure 1.1

```
        name: "Slice(1)"
        inputs: in, cin
        outputs: out, cout
        size: (50, 40)
    }

    group "CLB" {
        Slice0;
        Slice1;
    }

    wire SwitchMatrix.out0 -> Slice0.in {
        thickness: 2
    }

    wire SwitchMatrix.out1 -> Slice1.in {
        thickness: 2
    }

    wire Slice0.cout -> Slice1.cin {
        thickness: 1
    }

    layout {
        SwitchMatrix at (10, 10)
        Slice0 at (100, 10)
        Slice1 at (100, 70)
    }
}
```

Blockpic Language Specification

Blockpic is a domain-specific language (DSL) for engineering block diagram typesetting, designed to produce diagrams with the aesthetics of Xilinx UG474 technical documentation.

1. Overview

Blockpic focuses on clear, functional, and aesthetically minimalist diagrams. It prioritizes orthogonal routing, precise port placement, and a standard color palette for hardware logic primitives.

2. Core Concepts

2.1 Blocks and Primitives

A **Block** is the primary logical element. Each block has: - **Label**: A unique logical designation identifier (e.g. `AND1`) used for defining wire connections and layout references. - **Name**: An optional display name attribute (e.g. `name: "My Gate";`) shown inside the block shape on the diagram. If omitted, the block's **Label** is used for rendering. - **Kind**: The type of block (e.g., `AND`, `OR`, `FLIPFLOP`, `MUX`, `LUT`, `CUSTOM`). - **Input Pins**: Named input ports. - **Output Pins**: Named output ports. - **Parameters**: Key-value pairs for configuration (e.g., `init: 0x1234`). Supported parameters include: - `show_port_names`: Controls pin/port name visibility (fallback check). Set `show_port_names: "true"` to force rendering. - **Hint**: An optional block display hint. Supported values include: - `show-ports`: Forces rendering of input and output port names. - `hide-ports`: Forces hiding of input and output port names. - **Dimensions**: Optional width and height. - **Custom Rendering**: Support for custom rendering functions (Phase 2).

2.2 Pins (Ports)

Pins are the connection points on blocks. - **Type**: `input` or `output`. - **Position Hint**: `left`, `right`, `top`, `bottom`.

2.3 Wires

Wires connect an output pin of a source block to an input pin of a destination block. - **Source**: `(block_name, pin_name)`. - **Destination**: `(block_name, pin_name)`. - **Routing Hints**: Hints for the router (e.g., `straight`, `L-shape`, `Z-shape`, `feedback`). - **Signal Directionality**: Wires must render an arrowhead at the destination pin to clearly indicate the flow of the signal. - **Feedback Routing**: Support for backward-flowing signals (e.g., in Mealy/Moore state machines), routing wires around blocks to return to an earlier stage in the data path. - **Bus Attributes**: - **Thickness**: Specifies if the wire is a bus and its relative thickness. - **Crossline**: Supports a crossline indicator (often used with a number to indicate bus width).

2.4 Aesthetics (UG474 Style)

- **Colors**:
  - Primitives: Light Blue (`#E6F2FF`) with a Dark Grey (`#333333`) outline.
  - Containers: Light Grey (`#F2F2F2`) or transparent with dashed borders.
  - Special Paths: Light Orange (`#FFF2E6`) for carry chains or shift registers.
- **Typography**: Sans-serif (Helvetica or Arial), 8pt to 10pt.
- **Line Weights**: 0.5pt for wires, 1.5pt for buses.

2.5 Shape Library and IEEE Standards

Blockpic strictly adheres to **ANSI/IEEE Std 91-1984** for logic symbols. It supports both distinctive-shape and rectangular-shape symbols.

2.5.1 Basic Logic Gates

- **Distinctive Shapes**: Standard semi-circular (AND), pointed (OR), and triangle (NOT) shapes.
- **Rectangular Shapes**: Uniform rectangles using internal qualifying symbols:
  - `&` : AND
  - `>=1`: OR
  - `1` : Buffer/Inverter
  - `=1`: XOR
  - `X/Y`: Specialized functions.

2.5.2 Dependency Notation (for Complex Blocks) For MSI components like Registers and Counters, Blockpic supports IEEE dependency notation to represent internal logic relationships: - `G` **(AND)**: Gate dependency. - `C` **(Control)**: Timing/Clock dependency (often used with the `>` dynamic indicator for edge-triggering). - `S` **(Set) /** `R` **(Reset)**: Asynchronous set and reset. - `EN` **(Enable)**: Output enable. - `M` **(Mode)**: Mode dependency. - `V` **(OR)**: OR dependency.

2.5.3 Functional Indicators and Text Decorations

- **Bubble (`(o)`)**: External negation.
- **Wedge (`>`)**: Dynamic indicator for edge-triggered inputs.
- **Triangle (`(v)`)**: Three-state (Hi-Z) output indicator.
- **Overbar (`~text~`)**: Indicates an active-low signal (renders a line over the text).
- **Underbar (`_text_`)**: Text formatting indicator (renders a line under the text).

3. Syntax (Draft)

Blockpic uses a custom DSL designed for readability.

```
diagram MyLogic {
    config {
        style: "Xilinx";
        font: "Helvetica", 9pt;
    }

    // Block declarations
    block LUT6_A {
```

```
        kind: "LUT";
        params: { init: "0x8000000000000000" };
        inputs: I0, I1, I2, I3, I4, I5;
        outputs: O;
        size: (40, 60);
    }

    block DFF_1 {
        kind: "FLIPFLOP";
        inputs: D, CLK [bottom], CE [left];
        outputs: Q;
    }

    // Wire declarations
    wire LUT6_A.O -> DFF_1.D {
        hint: "Z-shape";
    }

    wire BUS_IN.O -> LUT6_A.I0 {
        thickness: 2;
        crossline: 8; // Indicates an 8-bit bus
    }

    // Optional Layout
    layout {
        LUT6_A at (10, 10);
        DFF_1 at (100, 10);
    }
}
```

4. Layout and Rendering

- **Logical First**: Connectivity is defined before layout.
- **Automatic Layout**: Default grid-based or flow-based layout (Phase 2).
- **Manual Refinement**: Deferred coordinate assignment via `layout` block.
- **Styles**: Support for multiple rendering styles (e.g., "Xilinx", "Generic").

5. Roadmap

Phase 1: Core Language and Renderer

- Define core DSL for blocks and wires.
- Basic `draw2d` rendering engine (rectangles, lines, text).
- Support for basic logic gates (`AND`, `OR`, `NOT`).
- Basic parameter and pin specification.

Phase 2: Advanced Features

- Implementation of the full IEEE 91-1984 shape library.
- Functional Dependency Notation support (`G`, `C`, `S`, `R`, `EN`).
- Complex MSI blocks (`REG`, `CTR`, `ALU`).
- Custom rendering functions for `CUSTOM` blocks.
- Basic auto-layout (grid layout).
- Improved wire routing (orthogonal).

Phase 3: Tooling

- CLI tool for rendering.
- Library of common hardware blocks.

5. Rendering Engine and Implementation

- **Implementation Language**: Go (Golang).
- **Rendering Library**: draw2d for 2D vector graphics.
- **Output Formats**: Export to SVG, PDF, and PNG via `draw2d` or complementary libraries.
- **Routing**: Automatic orthogonal routing with minimal crossings.
- **Precision**: LaTeX-like precision for text placement and port alignment.

6. Project Structure and Task Management

- **Specification**: Stored in `//ai/spec.md`.
- **Tasks**: Stored in `//ai/tasks/`.
- **Naming Convention**: All task files must start with a `NNN` prefix (e.g., `000_initial_lexer.md`), where `NNN` is a sequence number starting from `000`.

Task: Refine IEEE Graphics to Exact Standards (Phase 2)

Objective

Redo the element graphics to match the IEEE standard (ANSI/IEEE Std 91-1984) exactly as shown in the provided PDF document (https://wakerly.org/DDPP/DDPP3_pdf/IEEEsyms.pdf).

Requirements

- Review the current `shapes` package implementations for basic gates (AND, OR, NOT, XOR) and complex MSI blocks.
- Update the drawing algorithms to perfectly replicate the proportions, curve radii, and relative sizing shown in the official documentation.

- Specifically ensure that the rectangular notation for MSI blocks correctly sizes the "Common Control Block" and visually distinguishes it according to the standard.
- Ensure functional indicators (bubbles, wedges, triangles) are sized and positioned perfectly on the block boundaries.

Deliverables

- Updated `shapes` and `renderer` packages.
- Regenerated `shapes.svg`, `msi_blocks.svg`, and `dynamic_sizing.svg` demonstrating the refined, exact IEEE standards.

Task: Replicate figure_1_1 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_1_1.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_1_1.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_1_1, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_1 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_1.png`.

Status

- [x] Skipped. This image is a snippet of text, not a diagram.

Task: Replicate figure_2_2 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_2.png`.

Status

- [x] Skipped. This image is a table, not a diagram.

Task: Replicate figure_2_3 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_3.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_3.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_3, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_4 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_4.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_4.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_4, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_5 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_5.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_5.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_5, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_6 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_6.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_6.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_6, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_7 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_7.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_7.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).

- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_7, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_8 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_8.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_8.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_8, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_9 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_9.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_9.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_9, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_10 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_10.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_10.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_10, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_11 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_11.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_11.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_11, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_12 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_12.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_12.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_12, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_13 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_13.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_13.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_13, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_14 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_14.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_14.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).

- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_14, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_15 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_15.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_15.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_15, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_16 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_16.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_16.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_16, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_17 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_17.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_17.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_17, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_18 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_18.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_18.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_18, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_19 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_19.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_19.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_19, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure__2__20 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_20.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_20.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure__2__20, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure__2__21 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_21.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_21.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).

- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_21, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_22 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_22.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_22.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_22, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_23 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_23.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_23.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_23, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_2_24 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_2_24.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_2_24.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_2_24, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_3_1 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_3_1.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_3_1.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_3_1, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_3_2 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_3_2.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_3_2.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_3_2, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_3_3 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_3_3.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_3_3.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_3_3, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_3_4 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_3_4.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_3_4.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).

- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_3_4, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_3_5 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_3_5.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_3_5.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_3_5, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_4_1 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_4_1.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_4_1.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_4_1, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Task: Replicate figure_4_2 Rendering

Objective

Implement any necessary features in Blockpic and its SVG skin to perfectly replicate the rendering output of `third_party/xilinx/figure_4_2.png`.

Requirements

- Analyze the layout, connections, block shapes, and text placements in `third_party/xilinx/figure_4_2.png`.
- Identify any missing diagram features (e.g. specific block types, routing complexities, text alignments).
- Update the Blockpic diagram generator or the `ai/skin.svg` to support these features.
- Provide a side-by-side comparison in the documentation showing the original figure_4_2, the layout converted to Blockpic, and the Blockpic source code.

Deliverables

- A working Blockpic source file demonstrating the layout.
- The generated SVG that visually matches the Xilinx reference diagram as closely as possible.
- Documentation updates including the side-by-side comparison.

Workspace for tasks