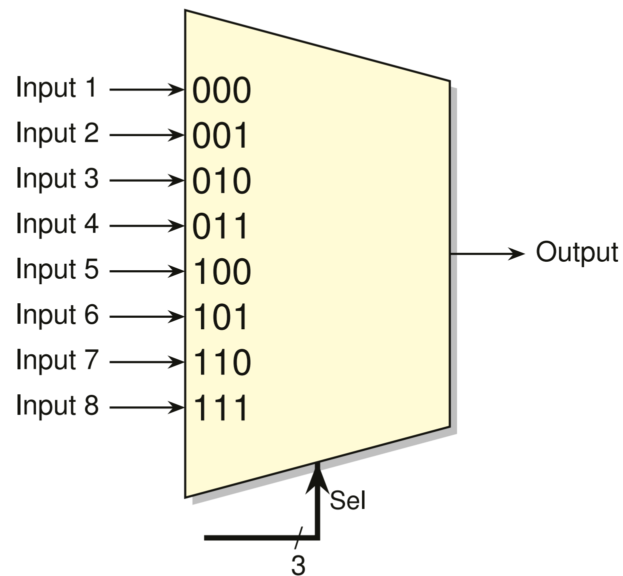
register-transfer-level

publication quality RTL diagrams



Lukas Rumpel

2026

under LPPL-1.3 license

Contents

1 Introduction **7**
1.1 Common Attributes and Sizing 8
1.1.1 Global Dimension Keys 8
1.1.2 The Anchor Convention 8

2 Memories **9**
2.1 Asynchronous Register 9
2.1.1 Description 9
2.1.2 Attributes 9
2.1.3 Anchors 9
2.1.4 Listing 9
2.2 Synchronous Register 10
2.2.1 Description 10
2.2.2 Attributes 11
2.2.3 Anchors 11
2.2.4 Listing 11
2.3 Register File 12
2.3.1 Description 12
2.3.2 Attributes 12
2.3.3 Anchors 12
2.3.4 Listing 12
2.4 D Flip Flop 13
2.4.1 Description 13
2.4.2 Attributes 13
2.4.3 Anchors 13
2.4.4 Listing 13
2.5 SR Flip Flop 14
2.5.1 Description 14
2.5.2 Attributes 14
2.5.3 Anchors 14
2.5.4 Listing 15
2.6 JK Flip Flop 15
2.6.1 Description 15
2.6.2 Attributes 15
2.6.3 Anchors 15
2.6.4 Listing 16

3 Structural Elements **17**
3.1 Custom Module (Black Box) 17
3.1.1 Description 17
3.1.2 Attributes 17

3.1.3 Anchors 17
3.1.4 Listing 17
3.2 Constant Value 18
3.2.1 Description 18
3.2.2 Attributes 18
3.2.3 Anchors 18
3.2.4 Listing 19
3.3 Cutset Decoration 19
3.3.1 Description 19
3.3.2 Attributes 19
3.3.3 Anchors 20
3.3.4 Listing 20

4 Combinatorial Logic **21**
4.1 Multiplexers and Demultiplexers 21
4.1.1 Description 21
4.1.2 Attributes 21
4.1.3 Anchors 21
4.1.4 Listing 21
4.2 Tri-State Buffer 22
4.2.1 Description 22
4.2.2 Anchors 23
4.2.3 Listing 23

5 Operators **25**
5.1 Arithmetical Operators 25
5.1.1 Description 25
5.1.2 Attributes 25
5.1.3 Anchors 25
5.1.4 Listing 25
5.2 Shifters 26
5.2.1 Description 26
5.2.2 Attributes 26
5.2.3 Anchors 26
5.2.4 Listing 26
5.3 Boolean Operations 27
5.3.1 Description 27
5.3.2 Attributes 27
5.3.3 Anchors 27
5.3.4 Listing 27

6 Signals, Routing and Signal Manipulation **29**
6.1 Signal Manipulation 29
6.1.1 Description 29

6.1.2 Attributes 29
6.1.3 Listing 29
6.2 Bus Styling 30
6.2.1 Description 30
6.2.2 Attributes 30
6.2.3 Listing 30
6.3 Advanced Routing (Z and U Routes) 31
6.3.1 Description 31
6.3.2 Parameters 31
6.3.3 Listing 31

7 Examples **33**
7.1 Simple Multiplexer 33
7.2 Accumulator 34

1 Introduction

At the time of this package's creation, no easy-to-use TikZ package existed for generating high-quality, publication-ready RTL diagrams. While RTL diagrams can certainly be drawn using standard TikZ or `circuitikz` elements, anyone who has attempted this can confirm that it is a tedious and highly inconvenient process.

During the preparation of a publication, I needed to draw an extensive RTL diagram of a complete DSP architecture. This necessity led to the creation of `register-transfer-level`. With `register-transfer-level`, creating modern, accurate, and high-quality RTL diagrams is done in the blink of an eye!

This package is intended primarily for FPGA engineers (I feel your pain!), ASIC designers, researchers, students, and basically anyone dealing with digital hardware visualization. In this documentation the elements of `register-transfer-level` and their usage are presented.

register-transfer-level offers:

- **Publication-Ready Aesthetics:** Designed to mimic the clean, authoritative look of classic VLSI / digital design / computer architecture text books. It features subtle grouping colors, drop shadows for multi-instance blocks, and crisp orthogonal routing, while minimizing the amount code.
- **Parametric Components:** Multiplexers can be inferred automatically; the geometry is handled by the package.
- **Smart Bus Routing:** The Anchor points of the elements are preserved for easier signal / bus routing. Buses can also be created automatically.
- **Hardware Semantics:** Includes standard representations for D-Flip-Flops (with clock/enable pins), Tri-State Buffers, ALUs, Shift Registers, and Register Files to name a few.

register-transfer-level has been successfully battle-tested in rigorous, multi-round peer-review environments (e.g., IEEE Transactions on Industrial Electronics). It is specifically optimized to:

- Unambiguously demonstrate the separation of data path and control logic.
- Satisfy even the most highly specific demands for hardware critical path visualization.
- Overwhelm any doubts regarding your architectural competence.

Please consider this package an open invitation to share your suggestions, feature requests, and feedback. `register-transfer-level` is intended to be a community-driven, continuously evolving tool dedicated to raising the standard of digital design visualization.

Disclaimer: While `register-transfer-level` guarantees pristine hardware diagrams, it unfortunately cannot protect you from reviewers nitpicking your textual formulations.

1.1 Common Attributes and Sizing

To ensure visual consistency across complex RTL diagrams, `register-transfer-level` provides a unified set of sizing attributes. While standard TikZ parameters remain functional, using the dedicated `rtl_*` keys is highly recommended to maintain a clean and semantic naming scheme throughout your source code.

1.1.1 Global Dimension Keys

The following attributes allow for precise control over the geometric properties of RTL elements:

- **`rtl_width / rtl_height`**: Overrides the default dimensions for rectangular blocks like registers (`rtl_reg`) and custom modules.
- **`rtl_mux_width / rtl_mux_height`**: Tailored specifically for the trapezoidal geometry of Multiplexers and Demultiplexers.
- **`rtl_size`**: Defines the diameter for circular arithmetical operators (e.g., adders or multipliers).

1.1.2 The Anchor Convention

Connectivity is the backbone of any RTL diagram. `register-transfer-level` follows a predictable, hyphenated naming convention to make signal routing intuitive:

```
{node_name}-{anchor_identifier}
```

For instance, a register named `RegA` provides anchors such as `RegA-CLK` or `RegA-D`. This systematic approach ensures that you spend less time looking up documentation and more time routing your data path.

2 Memories

2.1 Asynchronous Register

2.1.1 Description

The `rtl_reg` element serves as the fundamental building block for asynchronous memory components and simple transparent buffers. Based on the `rtl_base_box`, it provides a clean, stylized rectangular container for data storage. Since it is implemented as a TikZ `rectangle`, it inherits all standard geometric anchors. For complex routing and precise signal alignment, utilizing these native anchors is highly recommended.

2.1.2 Attributes

While standard TikZ styling (like `fill` or `thick`) can be applied directly, the following package-specific attributes allow for easy dimensioning:

Attribute	Effect
`rtl_width`	Sets the minimum width of the register (Default: 1.5cm).
`rtl_height`	Sets the minimum height of the register (Default: 1.8cm).

2.1.3 Anchors

- **Standard Anchors**: `north`, `south`, `east`, `west`, and all corner anchors (`north west`, etc.).
- **Custom Anchors**: Only standard rectangular anchors are provided.

2.1.4 Listing

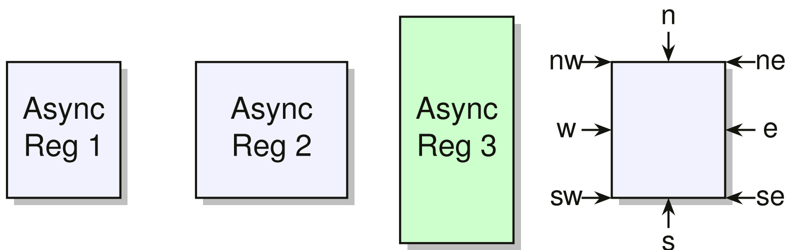
The following example demonstrates various sizing options and the available anchor points for signal routing.

Example: Async Reg/Buffer

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_reg] (Buffer1) at (0,0) {Async \\ Reg 1};
```

```
\node[rtl_reg, rtl_width=2cm, right=1cm of Buffer1]
    (Buffer2) {Async \\ Reg 2};
\node[rtl_reg, fill=green!20, rtl_height=3cm] (
   Buffer3) at (6.5,0) {Async \\ Reg 3};

\node[rtl_reg] (Buffer4) at (10,0) {};
\foreach \anchor/\label/\xshift/\yshift in {north/n
   /0/0.5, north west/nw/-0.5/0, west/w/-0.5/0,
   south west/sw/-0.5/0,
               north east/ne/0.5/0, east/e/0.5/0,
                  south east/se/0.5/0, south/s
                  /0/-0.5} {
               \draw[<-] (Buffer4.\anchor) -- ++(\
                  xshift, \yshift) node[pos=1.5]
                  {\label};
          }
\end{tikzpicture}
```



2.2 Synchronous Register

2.2.1 Description

The `rtl_reg_clk` element represents a standard edge-triggered memory component. It extends the asynchronous base model by integrating a dynamic clock indicator (a triangle) at the component's edge. This element is designed to signify synchronous behavior in data paths and provides specialized anchors to simplify clock tree integration.

2.2.2 Attributes

Attribute	Effect
rtl_width	Sets the minimum width of the register (Default: 1.5cm).
rtl_height	Sets the minimum height of the register (Default: 1.8cm).

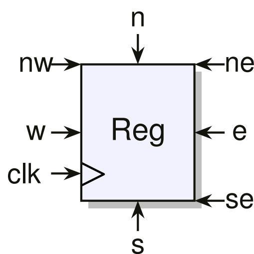
2.2.3 Anchors

- **Standard Anchors**: `north`, `south`, `east`, `west`, and all corner anchors.
- **Custom Clock Anchor**: `{node}-CLK`
  This anchor is precisely aligned with the tip of the clock indicator triangle, ensuring that clock signals meet the register at the correct logical point.

2.2.4 Listing

Example: Sync Reg/Buffer

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
        \node[rtl_reg_clk] (Reg) at (0,0) {Reg};
        \foreach \anchor/\label/\xshift/\yshift in {north/n
            /0/0.5, north west/nw/-0.5/0, west/w/-0.5/0,
                        north east/ne/0.5/0, east/e/0.5/0,
                           south east/se/0.5/0, south/s
                           /0/-0.5} {
                        \draw[<-] (Reg.\anchor) -- ++(\
                           xshift, \yshift) node[pos=1.5]
                           {\label};
                }
        \draw[<-] (Reg-CLK) -- ++(-0.5,0) node[left] {clk};
\end{tikzpicture}
```



2.3 Register File

2.3.1 Description

The `rtl_regfile` element is designed to represent memory arrays or register banks. To visually distinguish a file from a single register, this element utilizes a "double copy" shadow effect, simulating a stack of multiple memory instances. It maintains the standard rectangular anchor scheme while providing a visual representation of storage depth.

2.3.2 Attributes

Attribute	Effect
`rtl_width`	Sets the minimum width of the register (Default: 1.5cm).
`rtl_height`	Sets the minimum height of the register (Default: 1.8cm).

2.3.3 Anchors

- **Standard Anchors**: Full set of rectangular anchors (`north`, `south`, etc.).

2.3.4 Listing

Example: Register File

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
        \node[rtl_regfile] (file) at (0,0) {Reg File};
        \foreach \anchor/\label/\xshift/\yshift in {north/n
           /0/0.5, north west/nw/-0.5/0, west/w/-0.5/0,
                  north east/ne/0.5/0, east/e/0.5/0,
                     south east/se/0.5/0, south/s
                     /0/-0.5, south west/sw/-0.5/0} {
                  \draw[<-] (file.\anchor) -- ++(\
                     xshift, \yshift) node[pos=1.5]
                     {\label};
             }
\end{tikzpicture}
```

2.4 D Flip Flop

2.4.1 Description

The `rtl_dff` element implements a classic D-type flip-flop. To cater to different circuit requirements, the `register-transfer-level` package provides two variants: the standard `rtl_dff` and the `rtl_dff_inv`, which includes an additional inverted output ($\overline{\text{Q}}$). Both variants feature a clock input indicator and pre-defined internal pins.

2.4.2 Attributes

Attribute	Effect
`rtl_width`	Sets the minimum width (Default: 1.2cm).
`rtl_height`	Sets the minimum height (Default: 1.8cm).

2.4.3 Anchors

- **Specific Anchors**: `D`, `CLK`, `Q`, `Qbar` (inverted only).

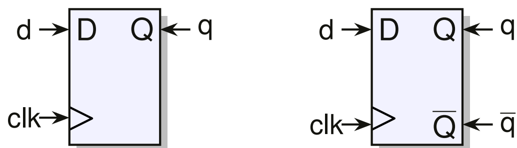
2.4.4 Listing

Example: D flip flop

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
        \node[rtl_dff] (dff1) at (0,0) {};
        \foreach \anchor/\label/\xshift/\yshift in {D/d
            /-0.5/0, CLK/clk/-0.5/0, Q/q/0.5/0} {
                        \draw[<-] (dff1-\anchor) -- ++(\
```

```
                                xshift, \yshift) node[pos=1.5]
                                {\label};
                }

            \node[rtl_dff_inv] (dff2) at (5,0) {};
            \foreach \anchor/\label/\xshift/\yshift in {D/d
               /-0.5/0, CLK/clk/-0.5/0, Q/q/0.5/0, Qbar/$\
               mathsf{\overline{q}}$/0.5/0} {
                             \draw[<-] (dff2-\anchor) -- ++(\
                                xshift, \yshift) node[pos=1.5]
                                {\label};
                }
      \end{tikzpicture}
```



2.5 SR Flip Flop

2.5.1 Description

The `rtl_sr_ff` represents a Set-Reset flip-flop. Unlike the D-type variants, it is implemented in a single configuration that features both complementary outputs (Q and $\overline{\mathsf{Q}}$) by default. It follows the standard rectangular footprint of the `rtl_base_box`.

2.5.2 Attributes

Attribute	Effect
`rtl_width`	Sets the minimum width (Default: 1.2cm).
`rtl_height`	Sets the minimum height (Default: 1.8cm).

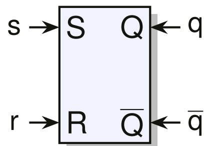
2.5.3 Anchors

- **Specific Anchors**: `S`, `R`, `Q`, `Qbar`.

2.5.4 Listing

Example: SR flip flop

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
        \node[rtl_sr_ff] (sr1) at (0,0) {};
        \foreach \anchor/\label/\xshift/\yshift in {R/r
           /-0.5/0, S/s/-0.5/0, Q/q/0.5/0, Qbar/$\mathsf{\
           overline{q}}$/0.5/0} {
                        \draw[<-] (sr1-\anchor) -- ++(\
                           xshift, \yshift) node[pos=1.5]
                           {\label};
                }
\end{tikzpicture}
```



2.6 JK Flip Flop

2.6.1 Description

The `rtl_jk_ff` element implements the versatile JK flip-flop. Designed for synchronous logic, it includes a clock indicator positioned between the J and K inputs. It provides a comprehensive set of specific anchors for all inputs and its complementary outputs.

2.6.2 Attributes

Attribute	Effect
`rtl_width`	Sets the minimum width (Default: 1.2cm).
`rtl_height`	Sets the minimum height (Default: 1.8cm).

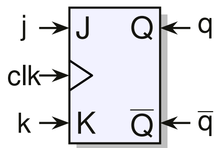
2.6.3 Anchors

- **Specific Anchors**: `J`, `CLK`, `K`, `Q`, `Qbar`.

2.6.4 Listing

Example: JK flip flop

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
        \node[rtl_jk_ff] (jk1) at (0,0) {};
        \foreach \anchor/\label/\xshift/\yshift in {J/j
           /-0.5/0, CLK/clk/-0.5/0, K/k/-0.5/0, Q/q/0.5/0,
           Qbar/$\mathsf{\overline{q}}$/0.5/0} {
                        \draw[<-] (jk1-\anchor) -- ++(\
                           xshift, \yshift) node[pos=1.5]
                           {\label};
                }
\end{tikzpicture}
```



3 Structural Elements

3.1 Custom Module (Black Box)

3.1.1 Description

The `rtl_module` element is designed to represent higher-level functional units or "black boxes" within a digital system. It is based on the `rtl_base_box` but features a distinct green tint and a dedicated label at the top. This element is ideal for hierarchical diagrams where the internal logic of a component is abstracted away. The label is automatically positioned at the `north` edge with a slight offset to prevent overlap with incoming signals.

3.1.2 Attributes

The module's name is passed as a mandatory argument to the style. Dimensioning follows the standard package scheme:

Attribute	Effect
`rtl_module={Name}`	Sets the identifying label at the top of the module.
`rtl_width`	Sets the minimum width (Default: 1.5cm).
`rtl_height`	Sets the minimum height (Default: 1.8cm).

3.1.3 Anchors

- **Standard Anchors**: `north`, `south`, `east`, `west`, and all corner anchors.
- **Note**: Use the `north` anchor with caution, as it shares space with the module label.

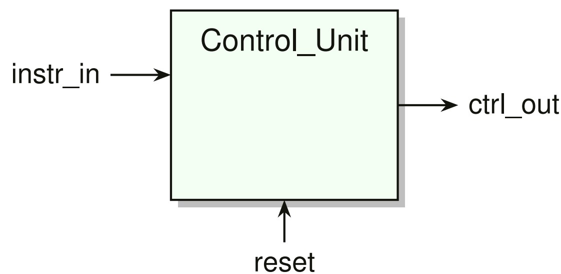
3.1.4 Listing

The following example demonstrates a scaled module with a custom name.

Example: Custom Module

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    % Module with specific name and dimensions
    \node[rtl_module={Control\_Unit}, rtl_width=3cm, rtl_height=2.5
        cm] (Mod1) at (0,0) {};
```

```
    % Connections
    \draw[<-] (Mod1.west) ++(0,0.5) -- ++(-1,0) node[left] {instr\_
       in};
    \draw[->] (Mod1.east) -- ++(1,0) node[right] {ctrl\_out};
    \draw[<-] (Mod1.south) -- ++(0,-0.7) node[below] {reset};
\end{tikzpicture}
```



3.2 Constant Value

3.2.1 Description

The `rtl_const` element is used to represent static signal values, parameters, or hardcoded addresses. Its distinct purple color serves as a visual cue that the signal originating from this block is non-volatile and fixed during the design's operation. Internally, it behaves like an asynchronous register without any clock or control logic.

3.2.2 Attributes

Attribute	Effect
`rtl_width`	Sets the minimum width (Default: 1.5cm).
`rtl_height`	Sets the minimum height (Default: 1.8cm).

3.2.3 Anchors

- **Standard Anchors**: Full set of rectangular anchors (`north`, `south`, etc.).

3.2.4 Listing

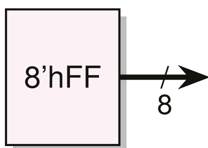
Constants are typically used to feed fixed values into operators or modules.

Example: Constant Value

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    % Constant value block
    \node[rtl_const] (C1) at (0,0) {8'hFF};

    % Example of signal origin
    \draw[->, rtl_bus={8}] (C1.east) -- ++(1.5,0) node[right] {};

\end{tikzpicture}
```



3.3 Cutset Decoration

3.3.1 Description

The `is_cutset` style is a specialized visual decorator used to mark timing boundaries, pipelining stages, or logic cuts within a diagram. Instead of being a standalone node, it is applied as an attribute to any existing rectangular element (like registers or modules).

The style automatically draws a dashed vertical line through the center of the component, extending slightly (5%) beyond the `north` and `south` boundaries. By utilizing the `foreground` PGF layer, the cutset remains clearly visible even when applied to filled or shaded elements.

3.3.2 Attributes

The style accepts an optional argument to customize the appearance of the dashed line, such as changing its color or thickness.

Attribute	Effect
`is_cutset={draw_options}`	Activates the cutset line. The argument allows standard TikZ draw options (Default: `dashed, thick`).

3.3.3 Anchors

- **Standard Anchors**: Full set of rectangular anchors (`north`, `south`, etc.).

3.3.4 Listing

The following example shows how to mark a timing boundary across a series of registers.

Example: Logic Cutset

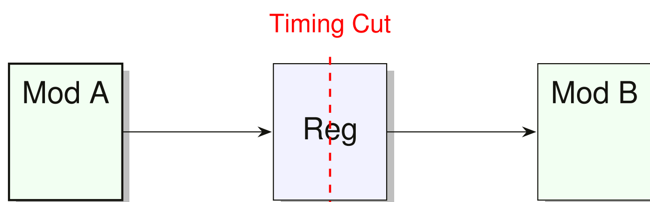
```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    % Sequence of registers
    \node[rtl_module={Mod A}] (M1) at (0,0) {};

    % Register with a red cutset line
    \node[rtl_reg, is_cutset={red}, right=2cm of M1] (R1) {Reg};

    \node[rtl_module={Mod B}, right=2cm of R1] (M2) {};

    % Signals
    \draw[->] (M1) -- (R1);
    \draw[->] (R1) -- (M2);

    % Label for the cutset
    \node[red, font=\sffamily\small] at (R1.north) [yshift=0.5cm] {
       Timing Cut};
\end{tikzpicture}
```



4 Combinatorial Logic

4.1 Multiplexers and Demultiplexers

4.1.1 Description

Multiplexers and Demultiplexers are the primary steering elements for data paths. In `register-transfer-level`, these are not simple nodes but are generated via the `\createMux` and `\createDemux` commands.

These commands automatically calculate the vertical distribution of ports based on the number of labels provided. Multiplexers (yellow-tinted trapezoids) converge multiple inputs into one output, while Demultiplexers diverge a single input into multiple outputs. Both elements feature a dedicated selection (`sel`) anchor at the bottom.

4.1.2 Attributes

The dimensions of the trapezoids can be adjusted using specific MUX/DEMUX sizing keys:

Attribute	Effect
`rtl_mux_width`	Sets the length of the input/output side (vertical span).
`rtl_mux_height`	Sets the thickness of the component (horizontal span).

4.1.3 Anchors

- **MUX Inputs**: `{name}-in-1` to `{name}-in-n`.
- **DEMUX Outputs**: `{name}-out-1` to `{name}-out-n`.
- **Common**: `{name}-sel` (bottom), `{name}-out` (MUX) or `{name}-in` (DEMUX).

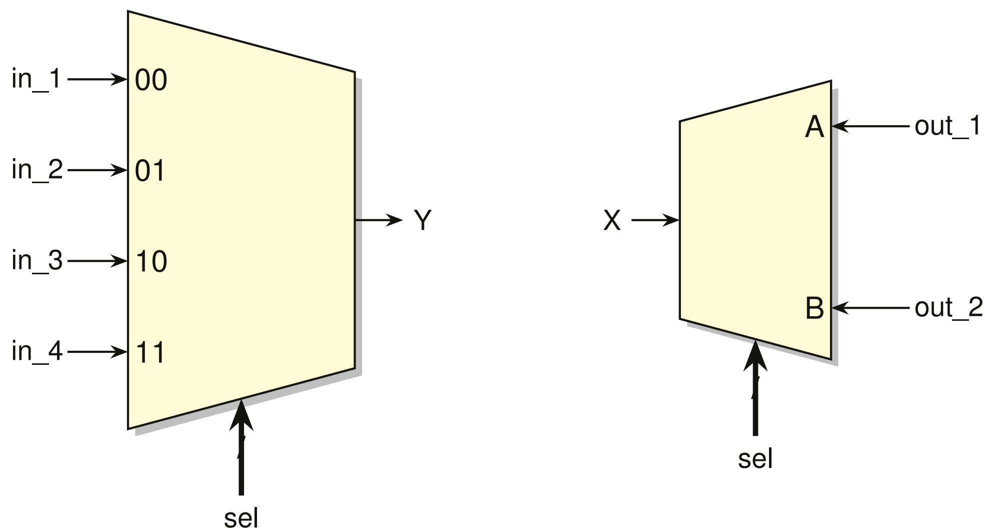
4.1.4 Listing

Example: MUX and DEMUX

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    % 4-to-1 MUX
    \createMux[rtl_mux_width=3cm]{M1}(0,0){00, 01, 10, 11}
```

```
    \draw[<-, rtl_bus={}] (M1-sel) -- ++(0,-1.6) node[below] {sel};
    \draw[->] (M1-out) -- ++(0.8,0) node[right] {Y};
    \foreach \anchor/\label/\xshift/\yshift in {1/in\_1/-1/0, 2/in
        \_2/-1/0, 3/in\_3/-1/0, 4/in\_4/-1/0} {
         \draw[<-] (M1-in-\anchor) -- ++(\xshift, \yshift) node[pos
             =1.5] {\label};
    }

    % 1-to-2 DEMUX
    \createDemux[rtl_mux_width=2cm]{D1}(8.5,0){A, B}
    \draw[<-] (D1-in) -- ++(-0.8,0) node[left] {X};
    \draw[<-, rtl_bus={}] (D1-sel) -- ++(0,-1.6) node[below] {sel};
    \foreach \anchor/\label/\xshift/\yshift in {1/out\_1/1.3/0, 2/
        out\_2/1.3/0} {
         \draw[<-] (D1-out-\anchor) -- ++(\xshift, \yshift) node[pos
             =1.5] {\label};
    }
\end{tikzpicture}
```



4.2 Tri-State Buffer

4.2.1 Description

The `rtl_tristate` element represents a controlled buffer used for bus arbitration. It features a triangular shape with an integrated `EN` (Enable) label at the top. This element is essential for modeling high-impedance states in shared signal lines.

4.2.2 Anchors

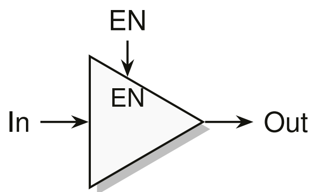
- **Specific Anchors**: Inherits `regular polygon` anchors. The `north` anchor is pre-configured for the Enable signal.

4.2.3 Listing

Example: Tri-State Buffer

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_tristate, rtl_size=2cm] (TS) at (0,0) {};

    \draw[<-] (TS.west) -- ++(-0.8,0) node[left] {In};
    \draw[->] (TS.east) -- ++(0.8,0) node[right] {Out};
    \draw[<-] (TS.north) -- ++(0,0.6) node[above] {EN};
\end{tikzpicture}
```



5 Operators

5.1 Arithmetical Operators

5.1.1 Description

The `rtl_op` element is used to represent mathematical operations within the data path, such as addition, subtraction, or multiplication. It features a circular shape with a clean white fill and a drop shadow, making it stand out as a functional node.

5.1.2 Attributes

The size of the operator circle can be adjusted to accommodate longer symbols or more complex arithmetic identifiers.

Attribute	Effect
`rtl_size`	Sets the diameter of the circular operator (Default: 0.8cm).

5.1.3 Anchors

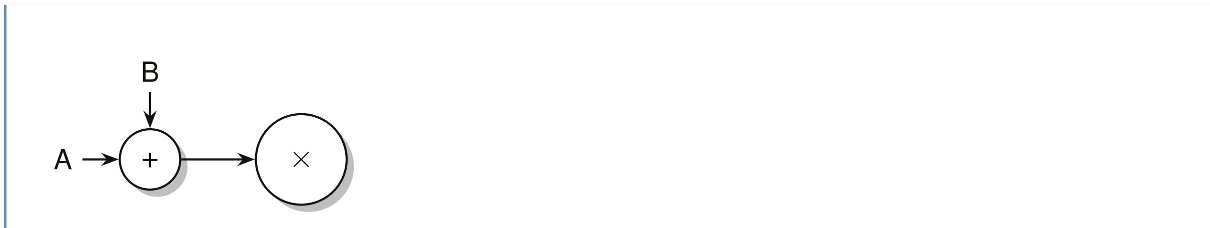
- **Standard Anchors**: `north`, `south`, `east`, `west`, and all degree-based anchors on the circle.

5.1.4 Listing

Example: Arithmetical Ops

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_op] (Add) at (0,0) {+};
    \node[rtl_op, rtl_size=1.2cm] (Mult) at (2.5,0) {$\times$};

    \draw[->] (Add) -- (Mult);
    \draw[<-] (Add.west) -- ++(-0.6,0) node[left] {A};
    \draw[<-] (Add.north) -- ++(0,0.6) node[above] {B};
\end{tikzpicture}
```



5.2 Shifters

5.2.1 Description

The rtl_shifter element represents bit-shifting operations (e.g., LSL, LSR, ASR). Visually, it is distinguished by a rectangular shape with rounded corners and a distinct orange tint, indicating a modification of the bus structure or bit positions.

5.2.2 Attributes

Attribute	Effect
rtl_width	Sets the minimum width (Default: 1.0cm).
rtl_height	Sets the minimum height (Default: 1.0cm).

5.2.3 Anchors

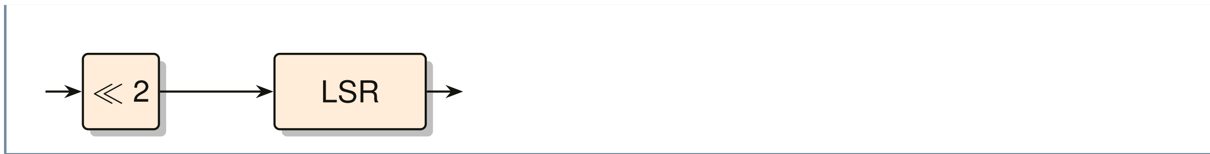
- **Standard Anchors**: All rectangular anchors (north, south, east, west, etc.).

5.2.4 Listing

Example: Shifter Element

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_shifter] (S1) at (0,0) {$\ll 2$};
    \node[rtl_shifter, rtl_width=2cm, right=1.5cm of S1] (S2) {LSR
       };

    \draw[->] (S1) -- (S2);
    \draw[<-] (S1.west) -- ++(-0.6,0);
    \draw[->] (S2.east) -- ++(0.6,0);
\end{tikzpicture}
```



5.3 Boolean Operations

5.3.1 Description

Complex logic gates or bitwise operations that do not fall into standard arithmetic or shifting categories are represented by the `rtl_boolean` style. These elements use a rounded rectangle with a violet background to signify logical transformation of signals.

5.3.2 Attributes

Attribute	Effect
`rtl_width`	Sets the minimum width (Default: 1.0cm).
`rtl_height`	Sets the minimum height (Default: 1.0cm).

5.3.3 Anchors

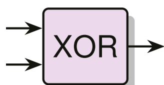
- **Standard Anchors**: Full set of rectangular anchors (`north`, `south`, etc.).

5.3.4 Listing

Example: Boolean Logic

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_boolean] (Log) at (0,0) {XOR};

    \draw[<-] (Log.west) ++(0,0.3) -- ++(-0.6,0);
    \draw[<-] (Log.west) ++(0,-0.3) -- ++(-0.6,0);
    \draw[->] (Log.east) -- ++(0.6,0);
\end{tikzpicture}
```



6 Signals, Routing and Signal Manipulation

6.1 Signal Manipulation

6.1.1 Description

The rtl_sigmanip element is used for non-arithmetic operations on signals, such as bit-slicing, concatenation, or sign extension. Its elliptical shape and soft orange tint visually separate it from functional logic blocks, indicating that the data is being reinterpreted or rearranged rather than computed.

6.1.2 Attributes

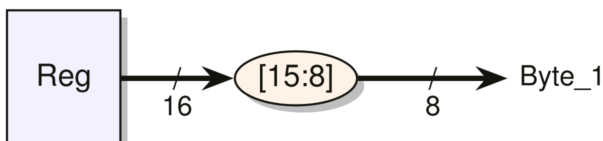
Attribute	Effect
inner sep	Adjusts the padding around the text (Default: 2pt).

6.1.3 Listing

Example: Signal Slicing

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_reg] (R1) at (0,0) {Reg};
    \node[rtl_sigmanip, right=1.5cm of R1] (Slice) {[15:8]};

    \draw[->, rtl_bus={16}] (R1) -- (Slice);
    \draw[->, rtl_bus={8}] (Slice) -- ++(3.5,0) node[right] {Byte
       \_1};
\end{tikzpicture}
```



6.2 Bus Styling

6.2.1 Description

To represent multi-bit data paths, `register-transfer-level` provides the `rtl_bus` decoration. Applying this style to a path increases the line width and adds a diagonal slash at the midpoint, accompanied by a label indicating the bus width.

A key feature of this decoration is its geometric intelligence: the element automatically calculates the total path length and places the / marker at exactly half the distance (`pos=0.5`). This ensures that the bus indicator remains perfectly centered, regardless of whether the path is a simple straight line or a complex route with multiple segments (e.g., a Z-route). Alternatively the position of the bus decoration can be set manually by using the `rtl_bus_flex` element.

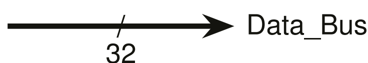
6.2.2 Attributes

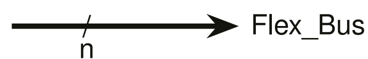
Attribute	Effect
`rtl_bus={width}`	Activates the bus decoration and sets the bit-width label.
`rtl_bus={width,pos}`	Activates the bus decoration, sets the bit-width label and alters the default positions of the decoration.

6.2.3 Listing

Example: Bus Decoration

```
\begin{tikzpicture}[>=Stealth, thick]
    \draw[->, rtl_bus={32}] (0,0) -- (3,0) node[right] {Data\_Bus};
    \draw[->, rtl_bus={}] (0,-1) -- (3,-1) node[right] {Generic\_
       Bus};
    \draw[->, rtl_bus_flex={n,0.33}] (7,0) -- (10,0) node[right] {
       Flex\_Bus};
\end{tikzpicture}
```





6.3 Advanced Routing (Z and U Routes)

6.3.1 Description

Manual routing with standard TikZ coordinates can become tedious in complex diagrams. `register-transfer-lev` introduces two specialized commands to automate common routing patterns:

- **\routeZ**: Creates a "Dogleg" or Z-shaped route. It moves horizontally, then vertically, and finally horizontal again to reach the destination.
- **\routeU**: Creates a "Feedback" or U-shaped route. This is ideal for signals that need to loop back from an output to a previous input stage.

6.3.2 Parameters

Command	Parameters
`\routeZ`	`[Style] {Start} {Kink_X} {End}`
`\routeU`	`[Style] {Start} {X_Space} {Y_Height} {End}`

6.3.3 Listing

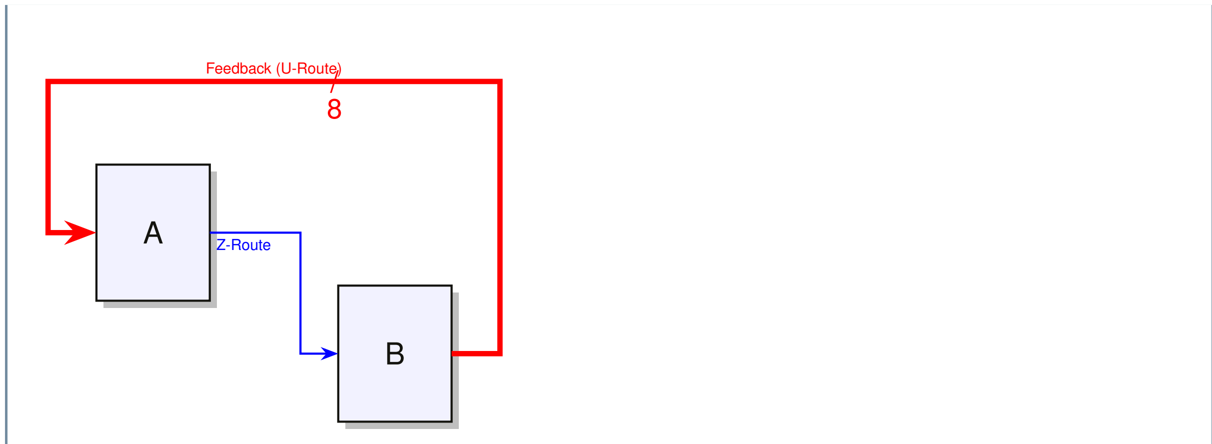
Example: Advanced Routing

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    \node[rtl_reg] (A) at (0,0) {A};
    \node[rtl_reg] (B) at (4,-2) {B};

    % Z-Route with 1.5cm horizontal segment
    \routeZ[->, blue]{A.east}{1.5}{B.west};

    % U-Route (Feedback) from B back to A
    \routeU[->, red, rtl_bus={8}]{B.east}{0.8}{4.5}{A.west};

    \node[blue, font=\tiny] at (1.5, -0.2) {Z-Route};
    \node[red, font=\tiny] at (2, 2.7) {Feedback (U-Route)};
\end{tikzpicture}
```

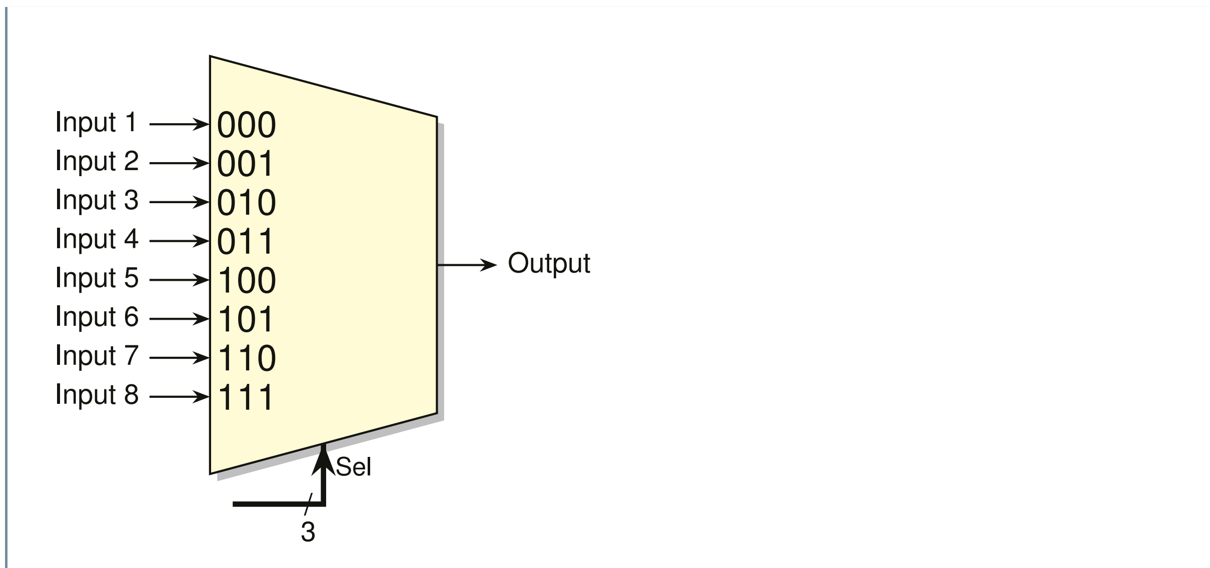

Feedback (U-Route)
8
A
Z-Route
B

7 Examples

7.1 Simple Multiplexer

Example: 8-to-1 Multiplexer

```
\begin{tikzpicture}[>=Stealth, thick,
scale=0.8]
    \createMux[rtl_height=3cm]{muxTitle}(0,0){{\Large 000}, {\Large
        001}, {\Large 010}, {\Large 011}, {\Large 100}, {\Large
       101}, {\Large 110}, {\Large 111}};
        \draw[<-] (muxTitle-in-1) -- ++(-1, 0) node[left] {Input
           1};
        \draw[<-] (muxTitle-in-2) -- ++(-1, 0) node[left] {Input
           2};
        \draw[<-] (muxTitle-in-3) -- ++(-1, 0) node[left] {Input
           3};
        \draw[<-] (muxTitle-in-4) -- ++(-1, 0) node[left] {Input
           4};
        \draw[<-] (muxTitle-in-5) -- ++(-1, 0) node[left] {Input
           5};
        \draw[<-] (muxTitle-in-6) -- ++(-1, 0) node[left] {Input
           6};
        \draw[<-] (muxTitle-in-7) -- ++(-1, 0) node[left] {Input
           7};
        \draw[<-] (muxTitle-in-8) -- ++(-1, 0) node[left] {Input
           8};
        \draw[->] (muxTitle-out) -- ++(1, 0) node[right] {Output};
    \draw[->, rtl_bus={3}, line width=2] ($(muxTitle-sel)$) --
       +(0,-1) node[right, font=\small, yshift=0.5cm] {Sel} --
       +(-1.5, -1)  (muxTitle-sel);
\end{tikzpicture}
```



7.2 Accumulator

Example: Accumulator Data Path

```
\begin{tikzpicture}[>=Stealth, thick, scale=0.8]
    % 1. Functional Elements
    \node[rtl_op] (Add) at (0,0) {+};
    \node[rtl_reg_clk, right=2cm of Add] (Reg) {Acc};

    % 2. Input Data Path
    \draw[<-, rtl_bus={16}] (Add.north) -- ++(0, 1.2) -- ++(-2, 0)
       node[left] {Data\_In};

    % 3. Forward Path (Adder to Register)
    \draw[->, rtl_bus={16}] (Add.east) -- (Reg.west) node[midway,
       above] {sum};

    % 4. Feedback Path (Register to Adder)
    % routeU: Start at east, move 1cm out, 2.5cm down, and end at
       Add.south
    \routeU[->, rtl_bus={16}, blue]{Reg.east}{1}{-2.5}{Add.west};

    % 5. Control Signals
    \draw[<-] (Reg-CLK) -- ++(-1, 0) node[below] {clk};

    % 6. Output
    \node[circ, right=0.75 of Reg.east] (n1){};
```

```
    \draw[->, rtl_bus={16}] (n1) -- ++(2,0) node[right] {Data\_Out
        };
\end{tikzpicture}
```