Figures That Follow the Design

Drawing Block Diagrams and Netlists with Blockpic in a Bazel Build

HDL Factory

Listing 1. The running example, `counter.v`.

```
module counter (
    input  wire       clk,
    input  wire       reset,
    output reg  [3:0] led
);
    always @(posedge clk) begin
        if (reset)
            led <= 4'b0;
        else
            led <= led + 4'b1;
    end
endmodule
```

***Abstract*—A figure in a design document is usually drawn once by hand, and then it quietly stops matching the design. This sample shows a way to stop that from happening. Blockpic draws block diagrams from short text files, and it draws gate-level schematics from the netlists that synthesis writes. A Bazel build runs synthesis, draws the figures and typesets the document in one graph. When the HDL changes, the next build redraws the figures. The running example is a 4-bit LED counter, synthesized by the open AMD 7-series flow of rules_openxc7.**

***Index Terms*—block diagrams, schematics, documentation, netlists, Yosys, Bazel, FPGA**

I. Introduction

EVERY design document needs pictures. A reader takes in a block diagram faster than three paragraphs that say the same thing. The trouble is that a picture is usually drawn once, in a drawing program, and saved as an image. The design keeps changing after that, and the picture does not. A year later, the figure shows a register that was removed and leaves out the one that replaced it.

Blockpic takes a different approach. A diagram is a short text file, kept in the repository beside the HDL it describes. The build draws the picture from that file every time. Blockpic can also draw a picture from the netlist that synthesis writes, so there is no file to forget at all. When the HDL changes, the next build redraws the figure from the new netlist.

This document is its own example. Both of its figures were drawn by the Bazel build that typeset it [7].

A. The running example

The example design is a 4-bit counter. It adds one to a register on every rising edge of `clk`, clears the register while `reset` is high, and shows the register on four LEDs. Listing 1 gives the whole design.

A sample document for blockpic, a language and tool that typesets digital block diagrams. Every figure in it is drawn during the build that makes this PDF: one from a hand-written diagram, and one from the netlist that Yosys synthesizes for an AMD 7-series part. The running example is a 4-bit counter that drives four LEDs.

Listing 2. The counter's architecture, `counter.blockpic`.

```
    block reset { kind: "INPUT_PIN"; outputs: out; }
    block one { kind: "INPUT_PIN"; outputs: out; }
    block zero { kind: "INPUT_PIN"; outputs: out; }
    block plus {
        kind: "ADD"; name: "+1";
        inputs: A, B; outputs: Y;
    }
    block clear {
        kind: "MUX"; name: "clear";
        inputs: A, B, S; outputs: Y;
    }
    block count {
        kind: "DFF"; name: "count";
        inputs: D, CLK; outputs: Q;
    }
    block led { kind: "OUTPUT_PIN"; inputs: in; }

    wire count.Q -> plus.A { thickness: 4; }
    wire one.out -> plus.B { thickness: 4; }
    wire plus.Y -> clear.A { thickness: 4; }
    wire zero.out -> clear.B { thickness: 4; }
    wire reset.out -> clear.S
    wire clear.Y -> count.D { thickness: 4; }
    wire clk.out -> count.CLK
    wire count.Q -> led.in { thickness: 4; }

    group "counter" { plus; clear; count }
}
```

The counter is small on purpose. Its schematic after synthesis still fits a column, and every cell in that schematic can be traced back to a line of the listing.

B. How to read this document

Section II draws the counter by hand, in the blockpic language. Section III draws the same counter from the netlist that synthesis writes for an AMD 7-series part. The build that ties the two to this PDF comes next (§IV). The document ends with what blockpic does not yet do well (§V).

II. Drawing a Block Diagram by Hand

An architecture diagram shows what a design does, not how synthesis built it. Someone has to decide what goes in it, so it is written by hand. In blockpic, it is written as text. Listing 2 is the counter's architecture.

A `block` has a label, a kind, and its input and output pins. The kind picks the symbol: `ADD` draws an adder, `MUX` a multiplexer and `DFF` a flip-flop. A `name`, such as `+1`, is drawn in place of the label. An `INPUT_PIN` or `OUTPUT_PIN` block is a port of the design. A `wire` joins an output pin to an input pin. A wire with a `thickness` is drawn as a bus: a thick line, crossed by a short slash with its width beside it. A wire can also name the type of the bus, as in `type: "nibble_t"`; the mark then shows the type in place of the width. A `group` draws a dashed box around blocks that belong together, here the counter itself. The group is laid out as one box, so no other block falls inside it.

Fig. 1 is the result. Blockpic placed every block and routed every wire; the file gives no coordinates. The style is that of the

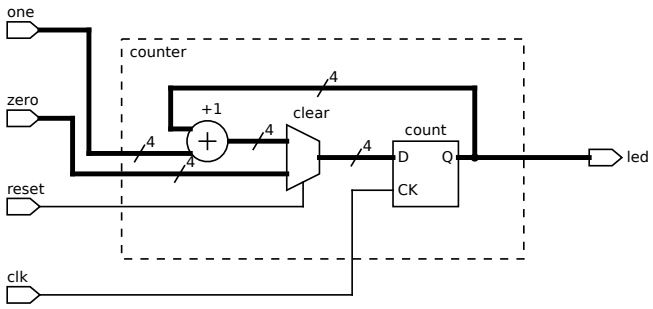

Fig. 1. The counter's architecture, drawn by blockpic from Listing 2. The thick lines, each marked 4, are the 4-bit buses.

figures in AMD's UG474 [6]: a plain sans-serif face, thin lines without arrowheads, and pin names such as `D` and `CK` inside the box. The feedback from the register to the adder is routed over the top, inside the group, and every 4-bit bus is drawn thick and marked with its width. The adder's inputs are equal, so blockpic gives the upper one to the wire that arrives from above, and no wire crosses another.

Guideline II.1 (Keep the diagram beside the code)**.** Put the `.blockpic` file in the same directory as the HDL it describes. A change to the design then shows the diagram in the same review, and the reviewer can ask for the two to agree.

Tip. Name blocks after the signals they make. Here the register is called `count`, as in the Verilog, so a reader can match the figure to the listing by name.

III. Drawing the Netlist

A hand-drawn diagram shows the intent. The netlist shows what the tools actually built. The two can differ in ways that matter: a register that synthesis removed, a bus that got wider, or a carry chain where an adder was expected. A drawing of the netlist shows those differences directly.

A. From HDL to a JSON netlist

The counter is synthesized by rules_openxc7 [5], which runs the open flow for AMD 7-series parts. Yosys [1] maps the design to the part's primitives with `synth_xilinx`. nextpnr [3] and Project X-Ray [4] then place and route it, but the drawing needs only the synthesis step. Its output is a JSON netlist: every cell, every port, and the nets between them, numbered bit by bit.

B. From the netlist to a drawing

The command `blockpic import` reads the netlist and writes a `.blockpic` file. From there, the netlist is drawn in exactly the same way as Fig. 1. Table I lists how each part of the netlist is drawn.

C. Reading the drawing

Fig. 2 is the counter as Yosys built it. Each part of Listing 1 can be found in it:

- The register is four `FDRE` flip-flops, one per bit. Each is drawn as a box with its pin names inside, as UG474 draws one. Their `R` pins are the synchronous reset, and `CE` is tied to `VCC`.
- The adder became one `CARRY4`, the carry chain of a 7-series slice [6]. Its `O_1` to `O_3` outputs feed the upper three flip-flops. Bit 0 needs no adder: an `INV` toggles it.
- Every port went through a buffer: `IBUF` on `clk` and `reset`, `OBUF` on each LED. The clock also goes through a `BUFG` onto the global clock network.
- The bus `led` was split into `led_0` to `led_3`, because each bit leaves through its own buffer.

None of this is in the hand-drawn Fig. 1, and none of it should be. The two figures answer different questions. The architecture says what the counter does. The netlist says what the FPGA will run.

Pitfall. A netlist drawing grows with the design. A counter fits a column, but a processor gives thousands of cells and an unreadable picture. Synthesis for the FPGA also flattens the design into one module. To draw one part of a large design, synthesize that part on its own, with it as the top level.

TABLE I
How `blockpic import` draws a netlist.

In the netlist	In the drawing
A cell, such as `FDRE`	A block of that kind
A module port	An input or output port block
A bus that goes whole to one pin	One thick wire, marked with its width, or its type when it has one
A bus whose bits go apart	A pin per bit, such as `O_1`
A constant 0 or 1	A `GND` or `VCC` tie-off at the pin
Any other constant	Its value, as a tie-off at the pin

Listing 3. The targets that draw Fig. 2.

```
vivado_project(
    name = "counter",
    srcs = ["counter.v"],
    part = "xc7a200tfbg484-2",
    top_level = "counter",
)

vivado_synthesis(
    name = "counter_synth",
    project = ":counter",
)

blockpic_netlist_diagram(
    name = "netlist",
    netlist = ":counter_synth",
    out = "netlist.svg",
)

svg_to_pdf(
    name = "netlist_pdf",
    src = ":netlist",
    out = "netlist.pdf",
)
```

IV. One Build from HDL to PDF

The figures stay current because nobody draws them. Bazel [7] builds them from their sources, like any other output. Fig. 3 shows the build graph behind this document.

Each box in the figure is one target, or one step inside a target. Listing 3 shows the targets in this document's `BUILD.bazel` that draw the netlist figure. `blockpic_netlist_diagram` takes any target whose only output is a Yosys JSON netlist. `vivado_synthesis` is one such target.

The document target lists `:netlist_pdf` among its inputs. So when `counter.v` changes, Bazel synthesizes the design again,

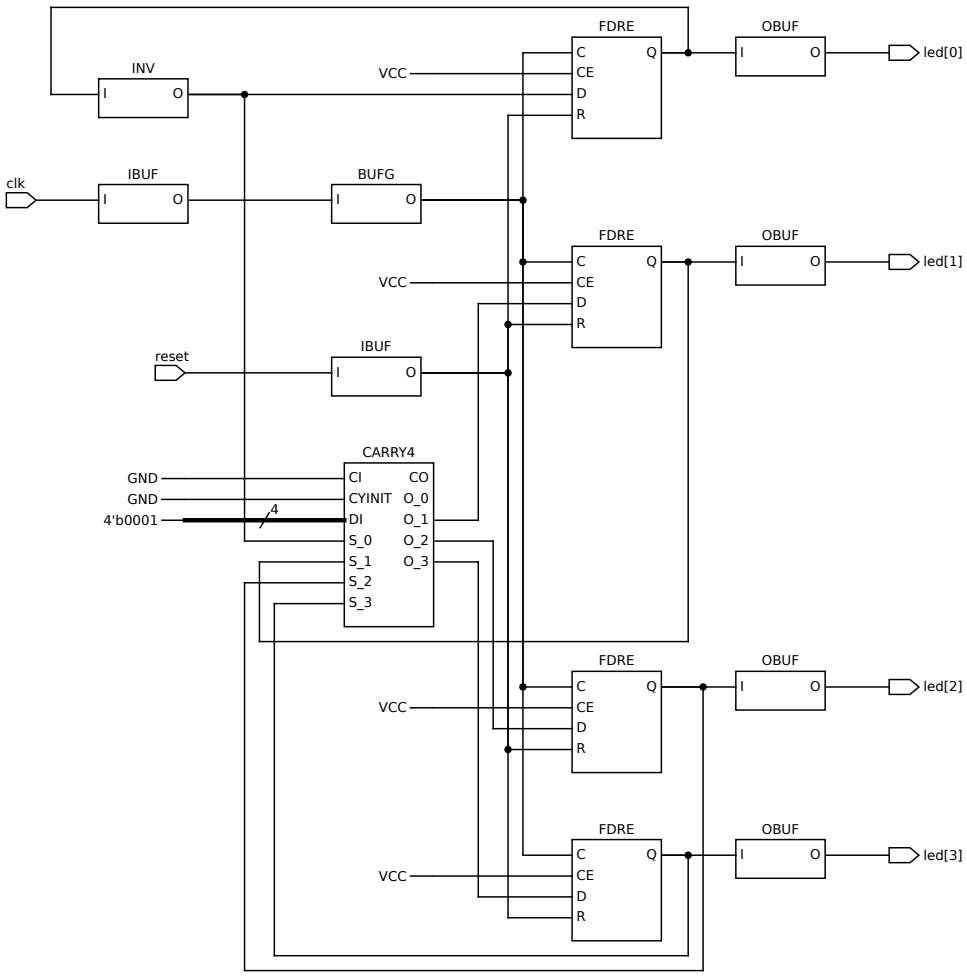

Fig. 2. The counter after synthesis for the xc7a200t, drawn by blockpic from the netlist that Yosys wrote.

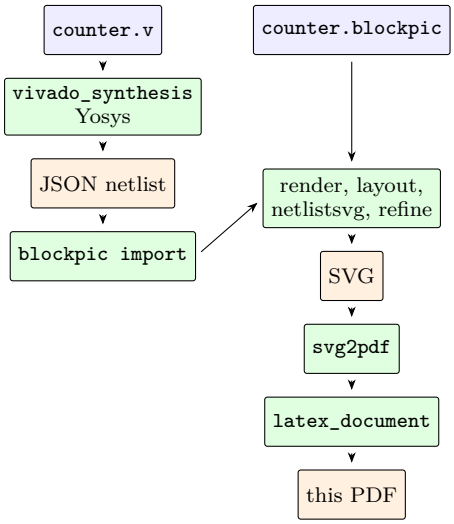

Fig. 3. The build graph of this document. Blue boxes are sources, green boxes are build steps, and orange boxes are their outputs. Both figures take the right-hand path.

redraws the figure and typesets the PDF again. When nothing changed, Bazel reuses all three results from its cache.

A. Turning SVG into PDF

pdflatex cannot include an SVG. `svg2pdf` converts each drawing into a one-page PDF of the same size, a pixel taken as a point. It embeds its font, DejaVu Sans, because a PDF for print must carry every font it uses. The converter is a short Node.js script, built from pinned npm packages, so the conversion is the same on every machine.

B. Publishing

A scheduled workflow builds this document and publishes it on the repository's release page. The same workflow attaches it to every tagged release of blockpic. A reader of a release therefore gets the figures drawn by that release's code.

Guideline IV.1 (Let the build own the figures)**.** Never commit a generated figure. Commit its source, the `.blockpic` file or the HDL, and let the build draw it. A committed image is one more file that can fall behind.

V. What Blockpic Does Not Do Yet

Blockpic is young, and both figures in this document show where.

- **Layout.** ELK places every block. It cannot be told to line the four flip-flops up in a column, which a person drawing

Fig. 2 would do. Wires that fan out, such as the clock and the reset, still bend on their way to each flip-flop.

- **Size.** A netlist of a few dozen cells is readable. One of a few thousand is not, so draw small modules (§III).

VI. Conclusion

A figure that the build draws cannot fall behind the design. This document showed two kinds. An architecture diagram is written by hand in a few lines of text and kept beside the HDL. A netlist drawing needs no source of its own, because synthesis writes it. Bazel ties both to the document, so a change to `counter.v` reaches Fig. 2 in the next build.

To start, write a `.blockpic` file for one block diagram in an existing document, and add the targets of Listing 3 to its build. The figure then stays correct for as long as the build passes.

References

[1] C. Wolf, "Yosys Open SYnthesis Suite," https://yosyshq.net/yosys/.
[2] N. Turley, "netlistsvg: draws an SVG schematic from a Yosys JSON netlist," https://github.com/nturley/netlistsvg.
[3] openXC7, "nextpnr for AMD 7-series FPGAs, the himbaechel Xilinx architecture," https://github.com/openXC7/nextpnr.
[4] F4PGA, "Project X-Ray: documenting the 7-series bitstream format," https://github.com/f4pga/prjxray.
[5] F. Filmar, "rules_openxc7: Bazel rules for AMD 7-series FPGAs with the open toolchain," https://github.com/filmil/bazel_rules_openxc7.
[6] AMD, "7 Series FPGAs Configurable Logic Block User Guide," UG474.
[7] The Bazel Authors, "Bazel," https://bazel.build.