FPGAs & PnR flows

98-154/18-224/18-624: Intro to Open-Source Chip Design

Field-Programmable Gate Arrays

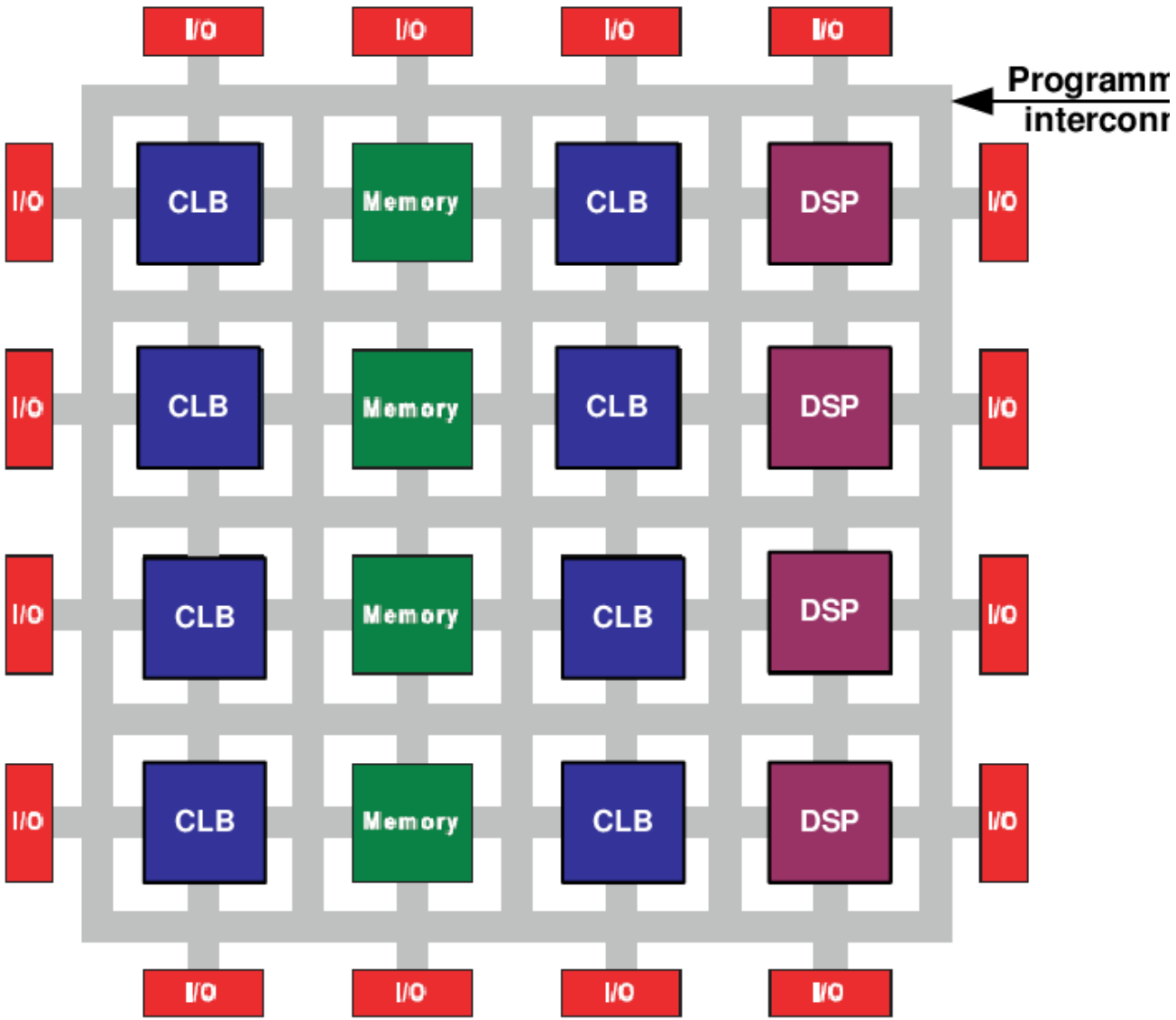
Basics

- FPGA = Field-Programmable Gate Array
- Design digital logic, test it within minutes/hours
  - As opposed to months/years to make a custom chip
- Range from tiny (~300 LEs) FPGAs used in embedded applications, to massive (~3M LEs) FPGAs used in datacenters

Why FPGA?

- Prototyping and emulation before fabricating a chip
- Video-processing, test equipment, software-defined radio
- Embedded (power-efficient) machine learning
- Virtual-reality, augmented-reality headsets
- High-frequency trading (financial markets)
- Digital-signal processing (cars, robots, satellites)
- Researchers constantly coming up with new applications

FPGA Overview



FPGA Internals: Overview

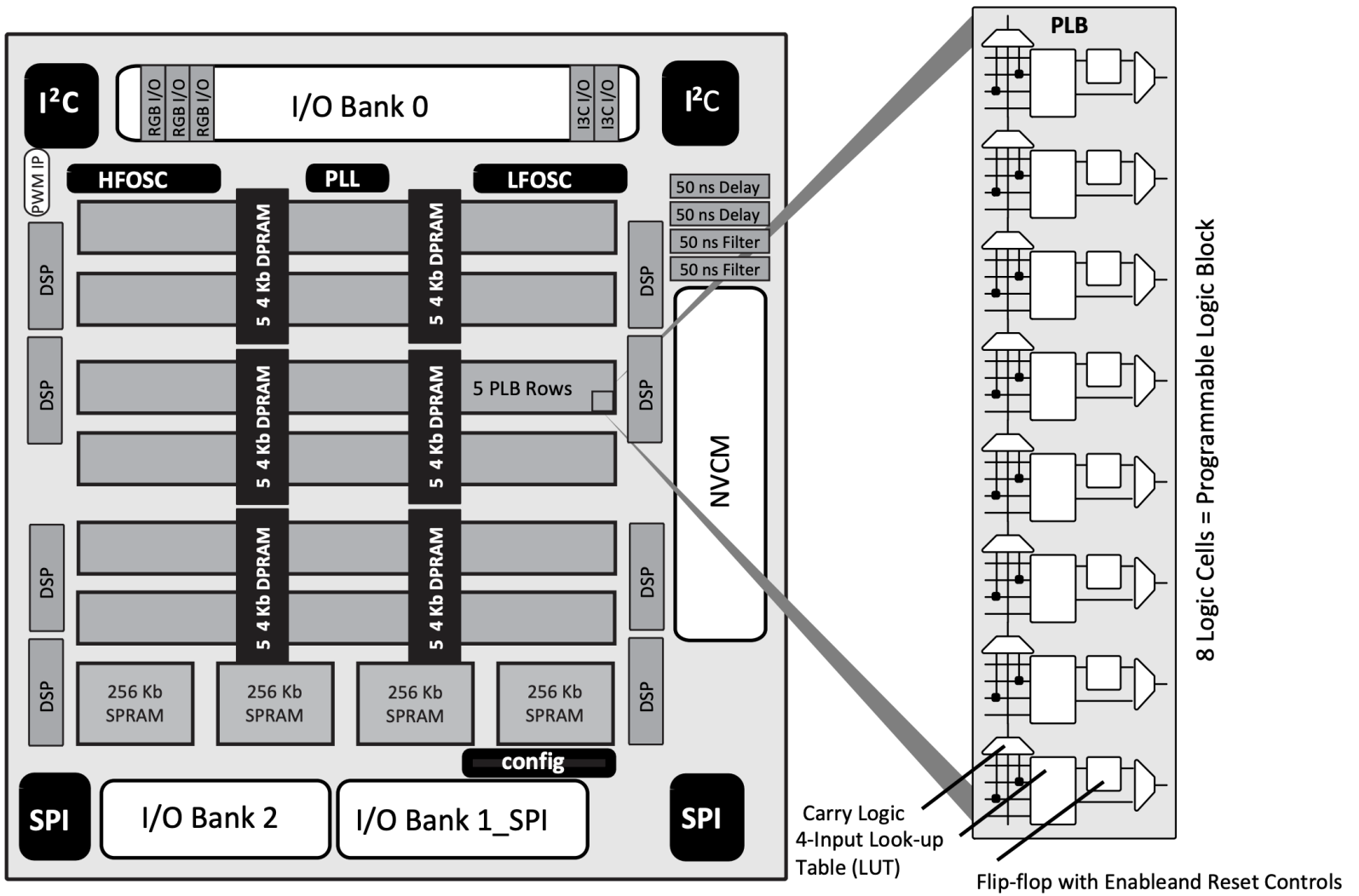


Figure 3.1. iCE40UP5K Device, Top View

image source: Lattice UltraPlus FPGA Datasheet

FPGA Internals: PLB/CLB

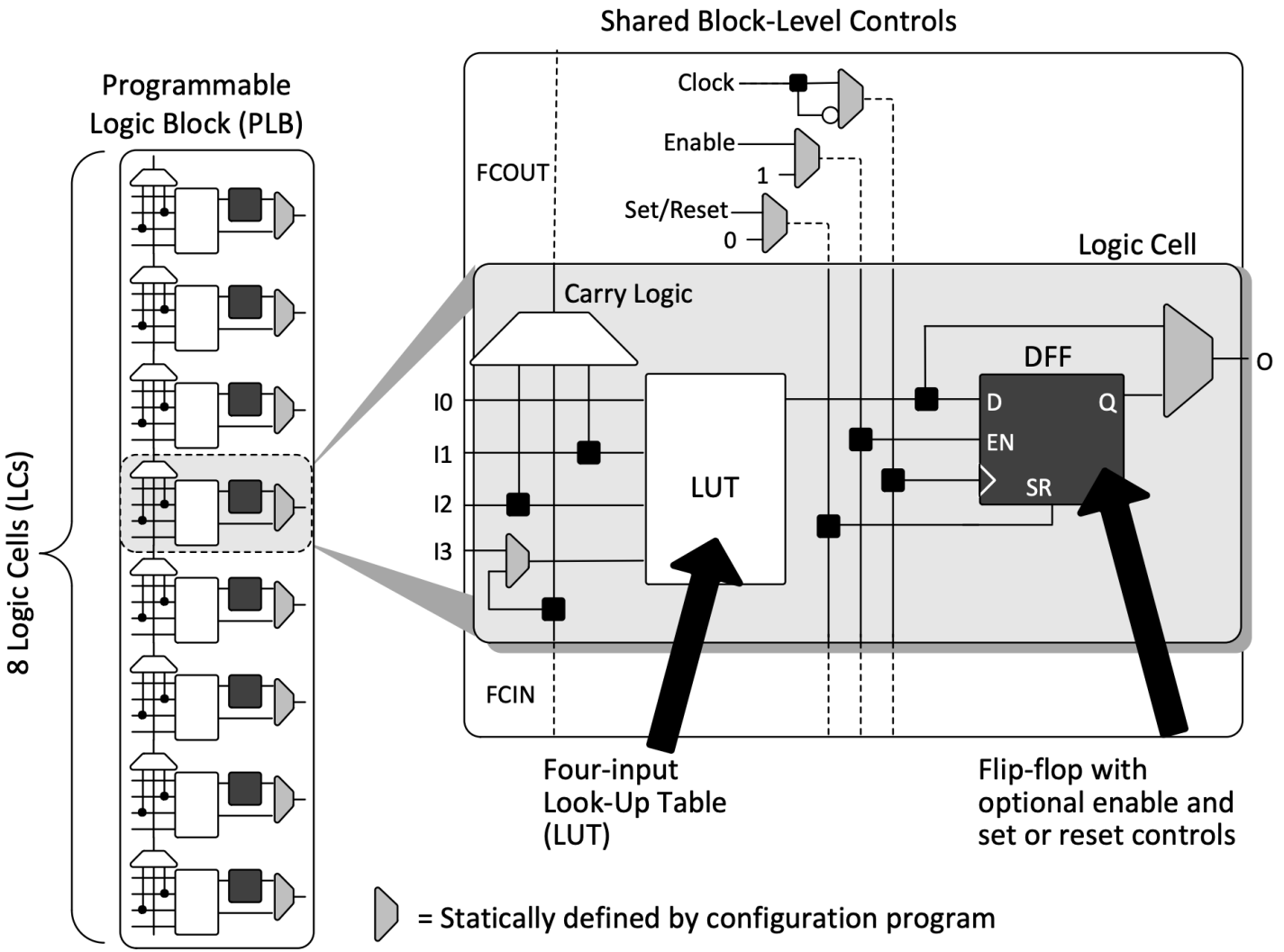


Figure 3.2. PLB Block Diagram

image source: Lattice UltraPlus FPGA Datasheet

FPGA Internals: Routing

Vendors are very cagey about the details of the routing-logic in their FPGAs, so we don't get nearly as much detail:

3.1.2. Routing

There are many resources provided in the iCE40 UltraPlus devices to route signals individually with related control signals. The routing resources consist of switching circuitry, buffers and metal interconnect (routing) segments.

The inter-PLB connections are made with three different types of routing resources: Adjacent (spans two PLBs), x4 (spans five PLBs) and x12 (spans thirteen PLBs). The Adjacent, x4 and x12 connections provide fast and efficient connections in the diagonal, horizontal and vertical directions.

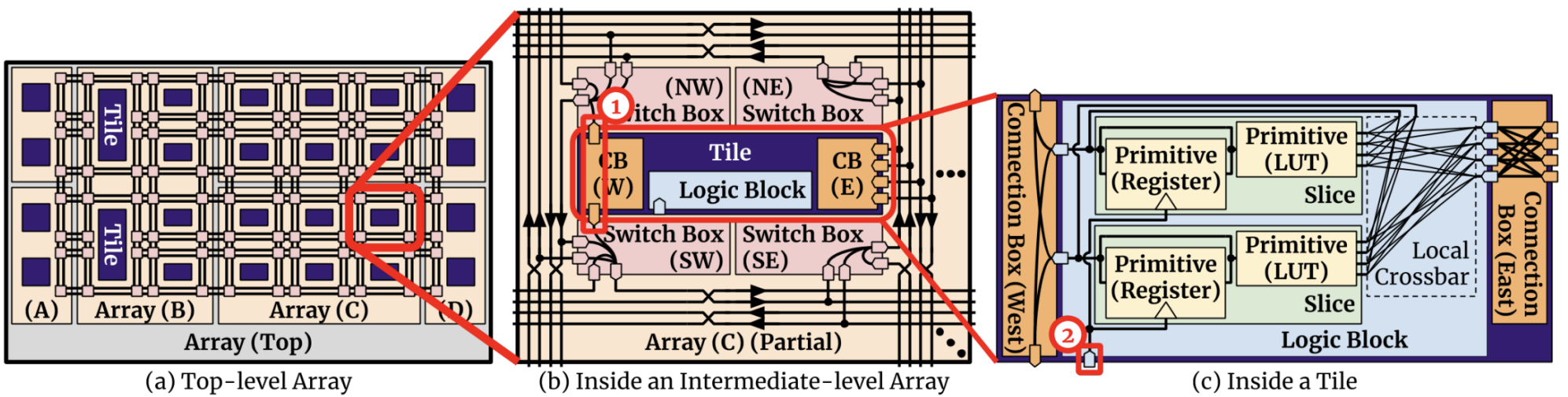
The design tool takes the output of the synthesis tool and places and routes the design.

image source: Lattice UltraPlus FPGA Datasheet

FPGA Internals: Routing

Fortunately, we have academia / open-source examples to study:

image credit: DOI 10.1145/3431920.3439294



(a) Top-level Array (b) Inside an Intermediate-level Array (c) Inside a Tile

Figure 2: FPGA architecture modeled by PRGA. The position, shape and size of the modules do not reflect the physical properties in an ASIC implementation. Programmable connections are shown as many-to-one connections in routing boxes and blocks. ① are bridging nets discussed in Sec. 3.3.3; ② is an unroutable clock pin directly connected to the global clock tree.

Interesting FPGA Families

- **Lattice iCE40**
- **Lattice ECP5**
- Xilinx 7-Series / UltraScale
- Altera (now Intel) Cyclone IV / V
- Efinix Trion
- many others…

Synthesis for FPGAs

Design for FPGAs

- Must be synthesizable (i.e. there must be a hardware equivalent)
- Reset signals: sync vs async, rising vs falling edge
- Dedicated clock trees: **use them**
- Limited resource congestion
- Manual floorplanning
- Constraints for I/O interfaces

IP Blocks

- “IP” = Intellectual Property, the hardware analog of a “library”
  - Can get open-source IPs from GitHub, OpenCores, etc.
- “soft IP” = Verilog/VHDL code (potentially encrypted)
- “hard IP” = dedicated logic inside the FPGA itself
  - Why hard IP?
  - Memories, arithmetic logic, transceivers, etc.

Hard IP: Memories

- Most applications require storing lots of data
- Registers / flip-flops are fast but there's relatively few of them
- Dedicated RAM blocks are slower but can store a lot more data

Hard IP: FPGA Memory Hierarchy

- Registers (flip-flops)
- Block RAMs
- URAM, SPRAM
- HBM, on-die DRAM
- External DRAM
- External Flash

Hard IP: DSP / Multipliers

- Multiplication takes a lot of logic, but is incredibly oft-used
- Dedicated DSP blocks implement multiplication in hardware (usually of a fixed width)
- Can chain together for arbitrary-width multiplication
- Digital Signal Processing, Machine Learning, Graphics, Control Systems, etc.

Hard IP: Xilinx UltraScale DSP48E2 (1)

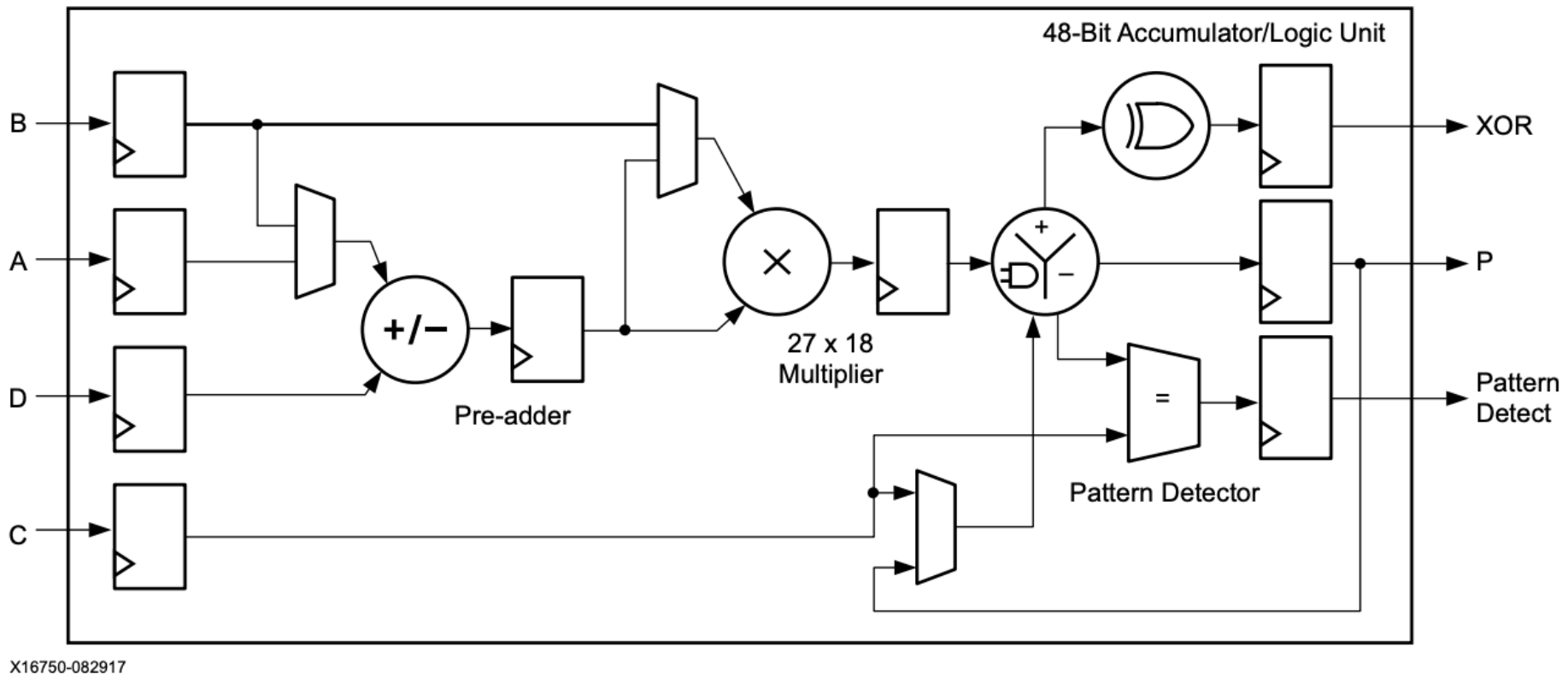


Figure 1-1: **Basic DSP48E2 Functionality**

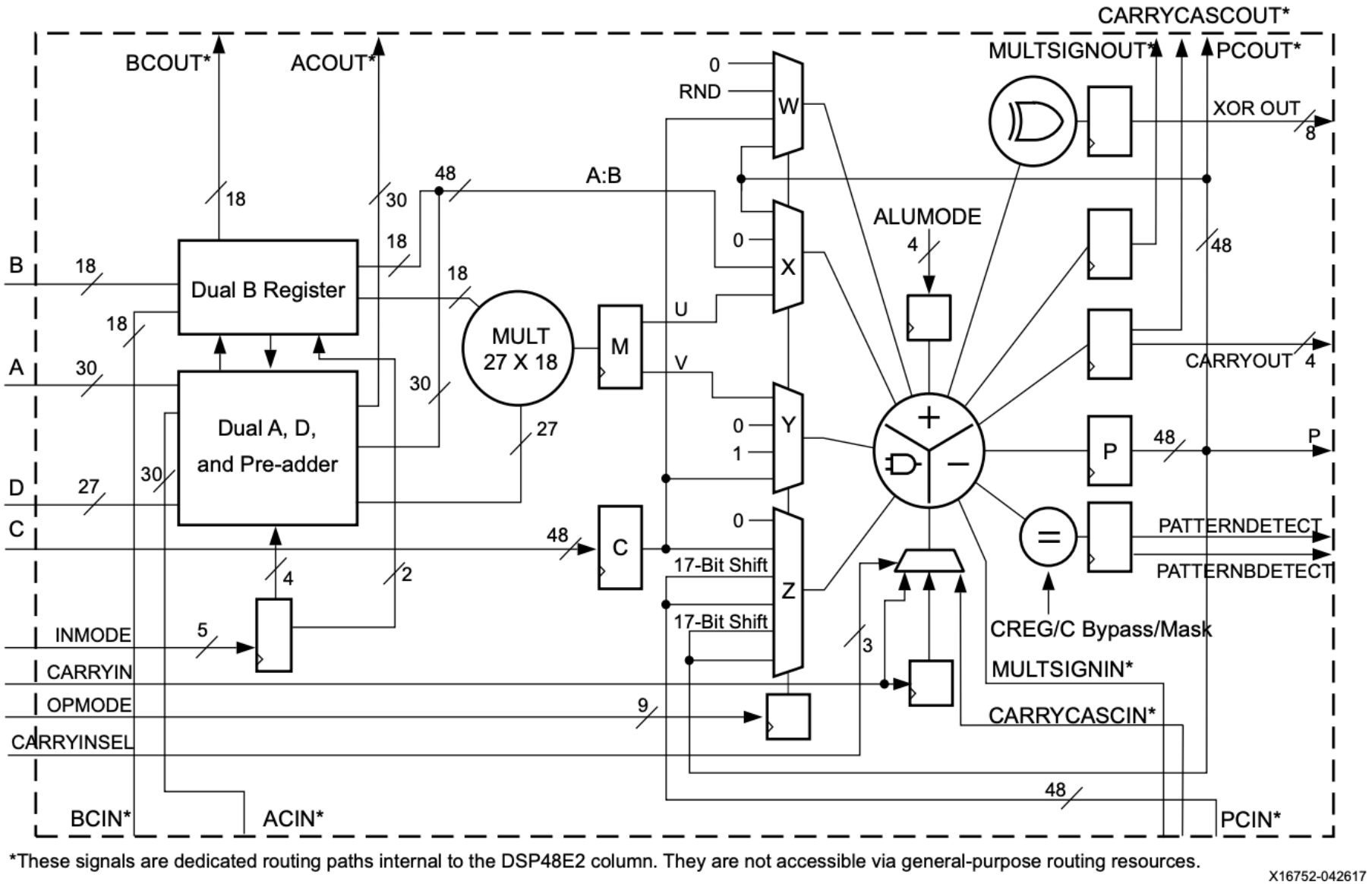


Figure 2-1: **Detailed DSP48E2 Functionality**

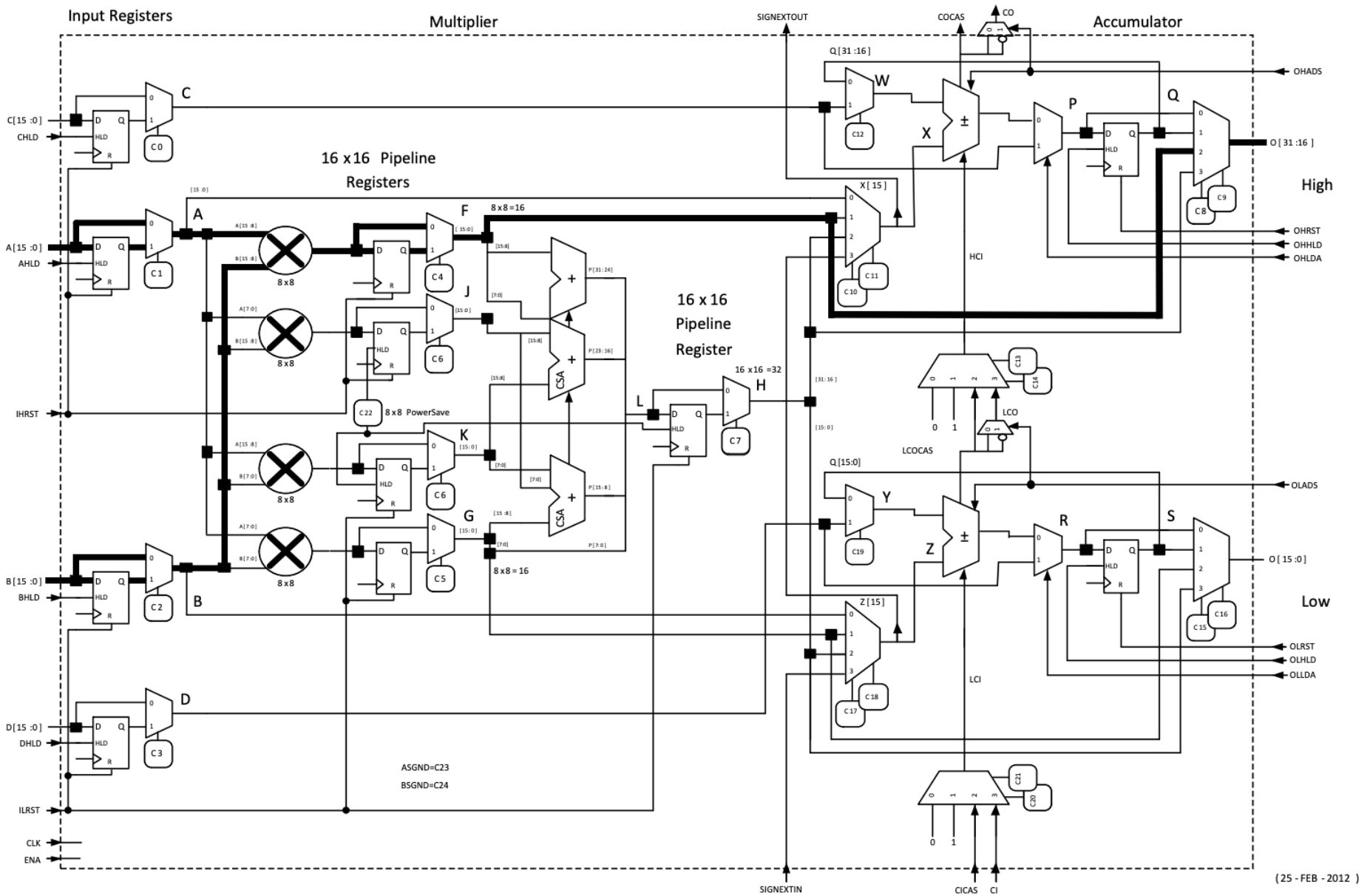


Figure 3.7. sysDSP 8-bit x 8-bit Multiplier

Other Hard IP

- Oscillators - generate clock frequency
- PLL - manipulate clock frequency and phase
- Transceivers - high speed interfaces like LVDS, PCIe, DDR3
- CPU cores - handle management, config, and other less-sensitive tasks

Inference vs Instantiation

- When you want to use hard IP, need to tell synthesis tool to use it
- Inference = describe **functionality** of what you want, let tool figure it out
- Instantiation = explicitly tell the tool **what IP block** you want it to use
- Both have different pros and cons

Inference Example: Multiplier

```
1 module ex_inf_mul (
2     input logic [7:0] a, b, c
3     output logic [16:0] d,
4     input logic clock
5 );
6     always_ff @(posedge clock) begin
7         d <= (a * b) + c;
8     end
9 endmodule
```

Inference Example: Memory

```
1  // Infer the RAM object
2  logic [31:0] ram[0:255];
3
4  always_ff @(posedge clock) begin
5      if (we) begin
6          ram[addr] <= data_in;
7      end
8      else begin
9          data_out <= ram[addr];
10     end
11 end
```

Instantiation Example: Multiplier

```
SB_MAC16 #(
    .TOPOUTPUT_SELECT(2'b11),
    .BOTOUTPUT_SELECT(2'b11),
    .PIPELINE_16x16_MULT_REG2(1'b1),
    .A_SIGNED(1'b0),
    .B_SIGNED(1'b0)
) m_multiplier (
    .A(a),
    .B(b),
    .C(16'h0000),
    .CE(1'h0),
    .CHOLD(1'h0),
    .CLK(1'h0),
    .O(c),
    ...
    ...
);
```

Instantiation Example: Memory

```
SB_SPRAM256KA ram_instance (
    .DATAIN(...),
    .ADDRESS(...),
    .MASKWREN(...),
    .WREN(...),
    .CHIPSELECT(...),
    .CLOCK(...),
    .STANDBY(...),
    .SLEEP(...),
    .POWEROFF(...),
    .DATAOUT(...)
);
```

Inference: Pros and Cons

- Pro: easy to write, very readable code
- Pro: mostly portable across synthesis flows
- Pro: easy to use with simulation and verification tools
- Con: Must match synthesis tool’s expectation closely
- Con: Edge cases can be tricky

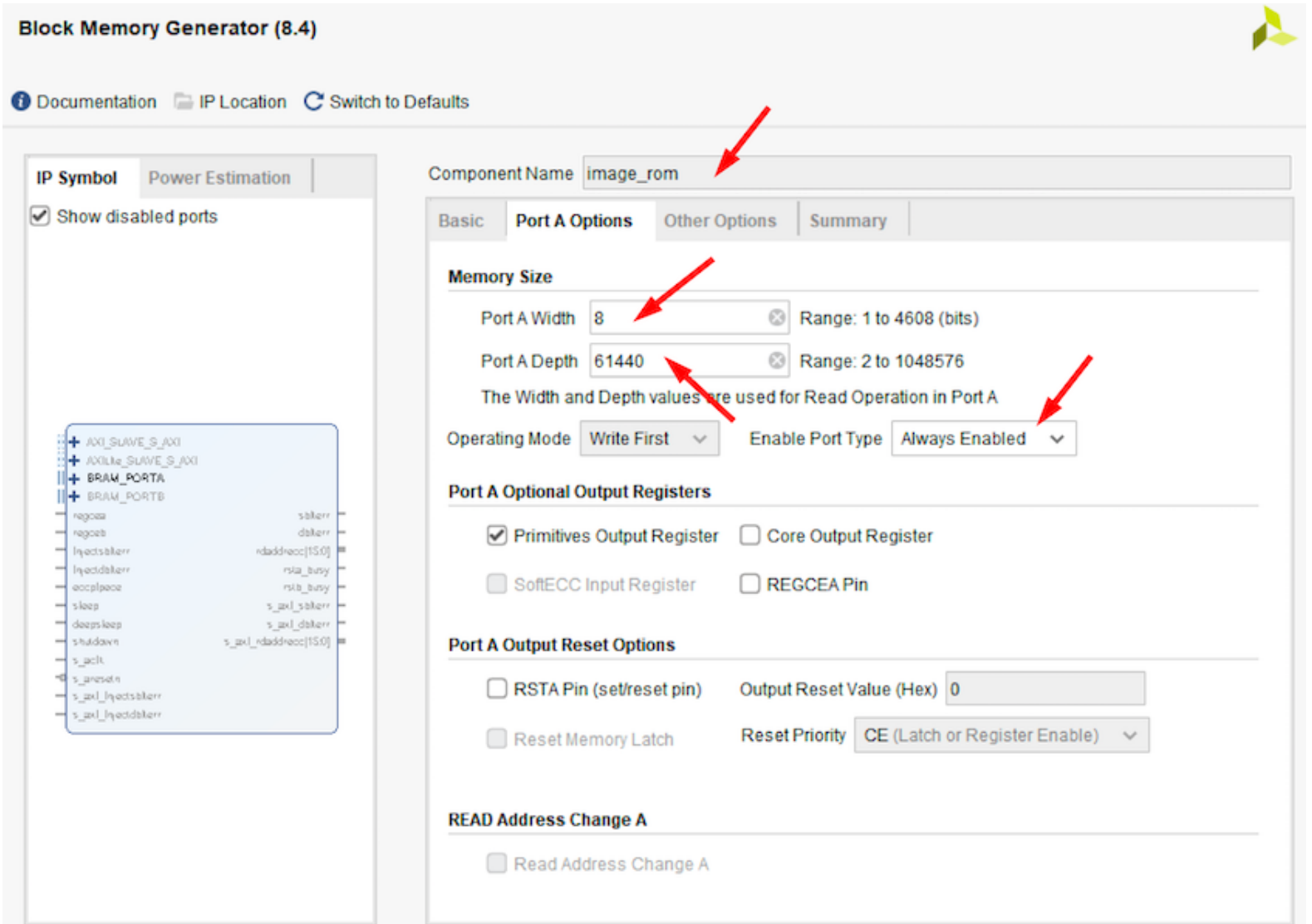
Instantiation: Pros and Cons

- Pro: Complete control over how the hardware in the FPGA is used
- Pro: Ability to use all features of the hard IP
- Con: Annoying to simulate and verify
  - Vendors usually have simulable models though
- Con: Less portable across platforms and FPGA vendors
  - Can build abstractions to reduce this complexity
- Con: Must read datasheet and manually figure out all the details

Instantiation Tools

- For particularly complicated instantiations, vendors will provide tools to make it easier (sometimes GUI, sometimes command-line)
- i.e. `ecppll` for ECP5 clocking config

Instantiation Tools: Xilinx Vivado



Demo: Inference with Yosys

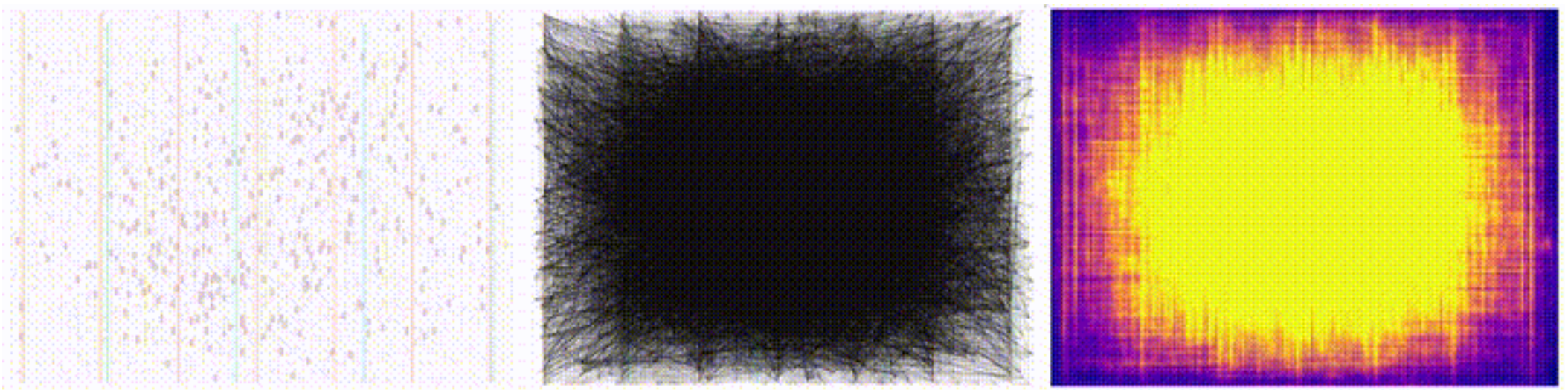
Demo: `ecppll` instantiation

FPGA PnR Flows

- Start with a “netlist” (list of LUTs and interconnections) from synthesis
- Placement: Place logic elements onto the FPGA
  - Optimize over Half-Perimeter Wavelengths
- Routing: Connect them together using the routing resources and multiplexers available on the FPGA, meet timing constraints
  - Optimize over wire-length and timing
- Bitstream Gen: Pack together config bits to a file that can be loaded to FPGA

FPGA PnR Flows

Placement, Routing, Utilization

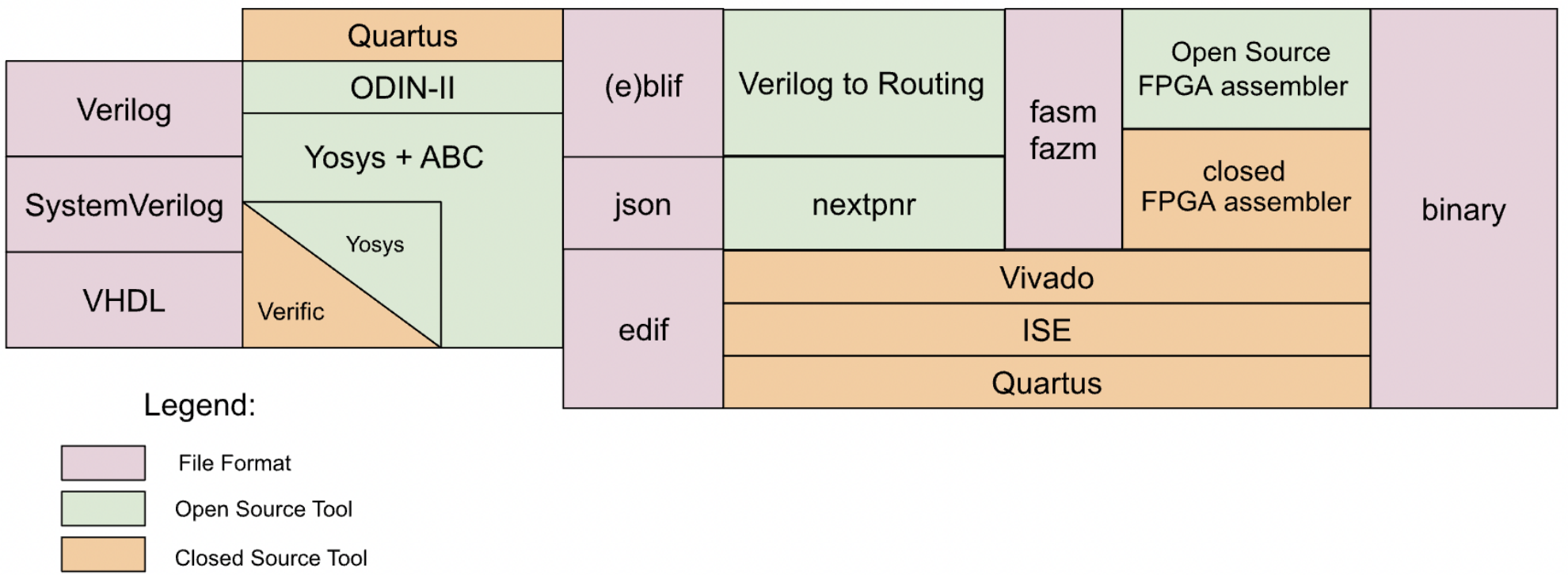


Open-Source PnR Flows

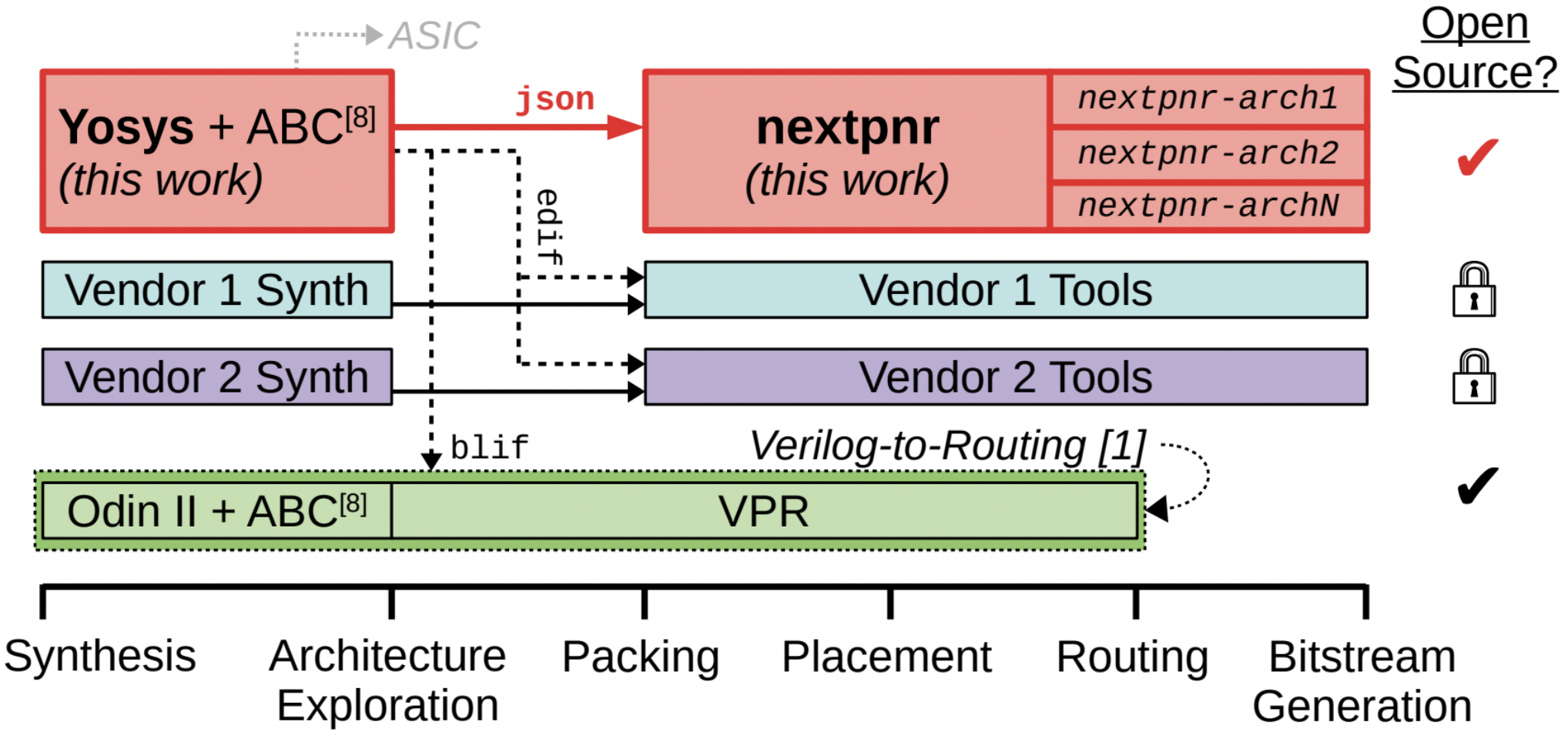
- Verilog to Routing / VPR
  - Focused on FPGA architecture research
- Yosys+NextPNR
  - Focused on strong routing and featureset
- F4PGA/SymbiFlow
  - Focused on standardization and flexibility

F4PGA

Focused on encompassing a large set of tools and standardizing data formats. Can use NextPNR or VPR under the hood.

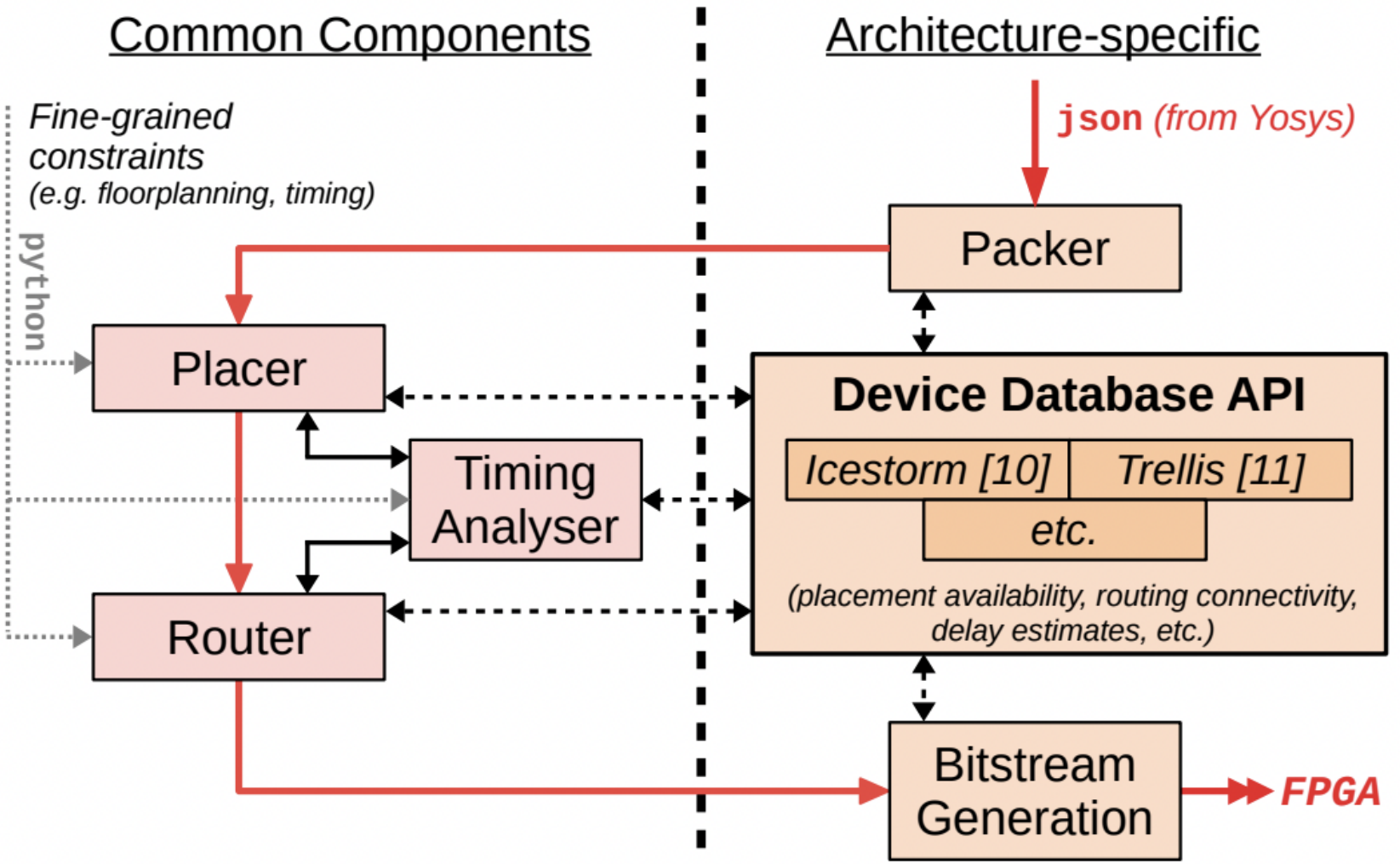


NextPNR: Overview



NextPNR paper

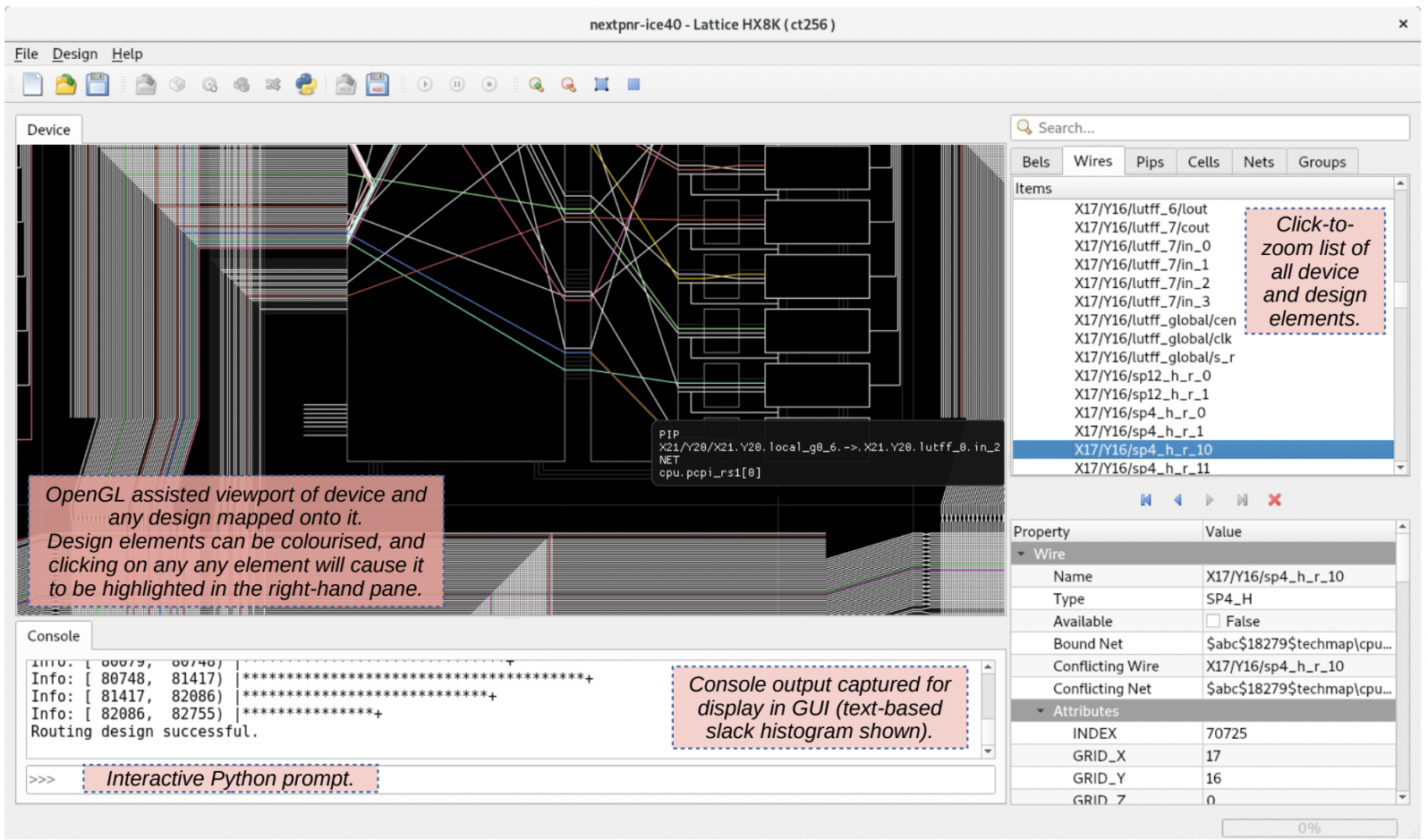
NextPNR: Architecture



NextPNR paper

NextPNR: Visualization

NextPNR is meant to be used entirely headless, thru Makefiles and scripts; but also has a GUI for visualizing results



NextPNR: Extensibility

- Case study on extending NextPNR to generate slowest possible routes
- Required only 87 lines of code to be added to NextPNR source

PHYSICALLY MEASURED PATH DELAY T_{delay} (IN MICROSECONDS) AND USED FRACTION OF ALL REACHABLE ROUTING WIRES F_{wires}.

Objective		Allowed routing runtime ~0s	<10s	<30s	<300s	<3000s
Shortest	T_{delay}	≤0.01				
	F_{wires}	0.0004				
Longest	T_{delay}	-	18.47	21.21	25.21	25.42
	F_{wires}	-	0.358	0.391	0.445	0.451

NextPNR: Further Reading

- GitHub
- Constraints
- Python Scripting
- NextPNR paper
  - Short paper, pretty good read
- Analytical Placement paper
  - Relatively approachable to read

Demo: FPGA Flow

Demo: Placement Constraints