DUHDe 2019 – Project Trellis

Open Tools for the ECP5 FPGA (and beyond)

David Shah
@fpga_dave
Symbiotic EDA

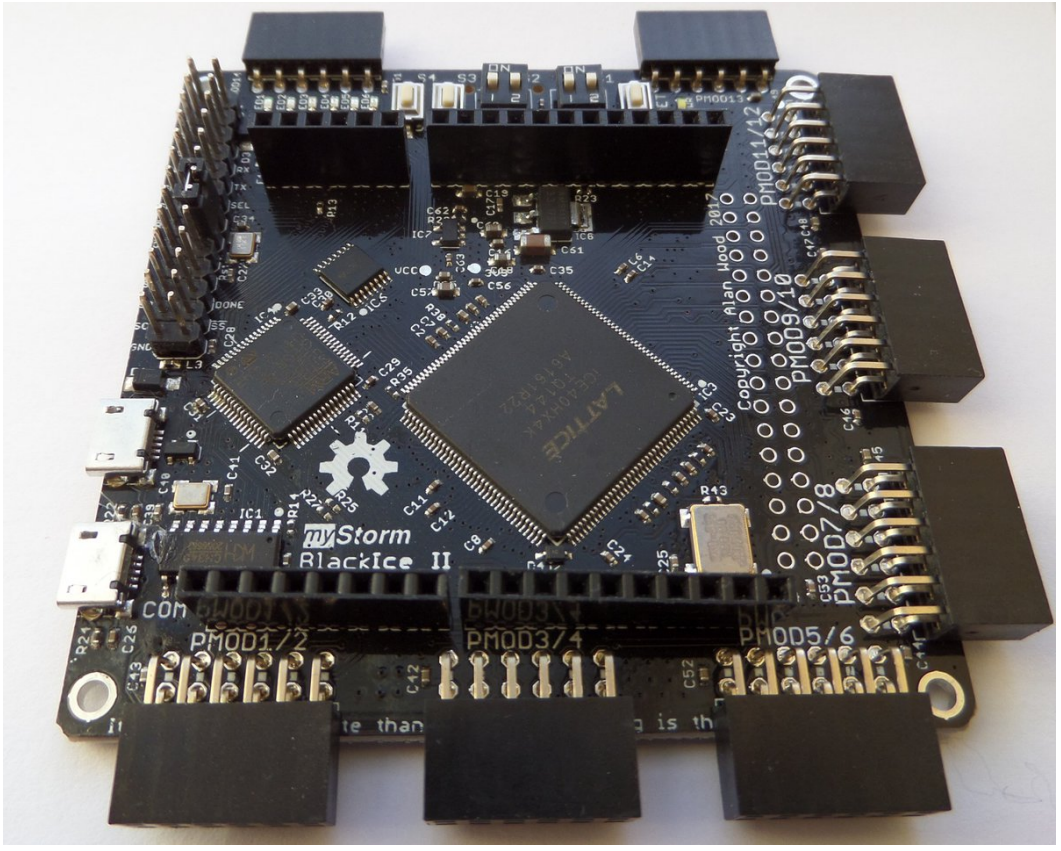
Slides: fpga.dev/duhde19.pdf

Symbiotic EDA

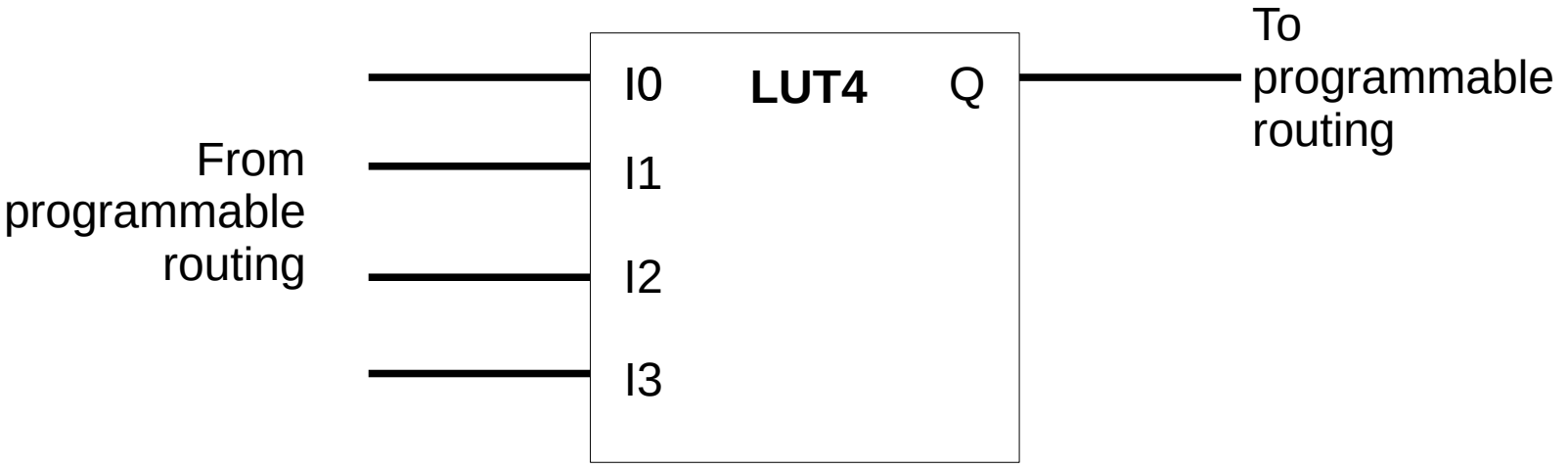
FPGA?

- Field Programmable Gate Array
- Chip containing **user-programmable** digital logic
- Can be (re)programmed “in the field” - older devices programmed once by literally burning fuses

FPGA?


myStorm
BlackIce II
PMOD1/2
PMOD3/4
PMOD5/6
PMOD7/8
PMOD9/10
PMOD11/12
Copyright Alan Wood 2017
LATTICE

FPGA Logic



Look Up Table (LUT) – basic logic element of an FPGA

a K-input LUT has 2^K bits of memory storing its **function**
e.g. 4-input AND: `1000 0000 0000 0000`
4-input OR: `1111 1111 1111 1110`

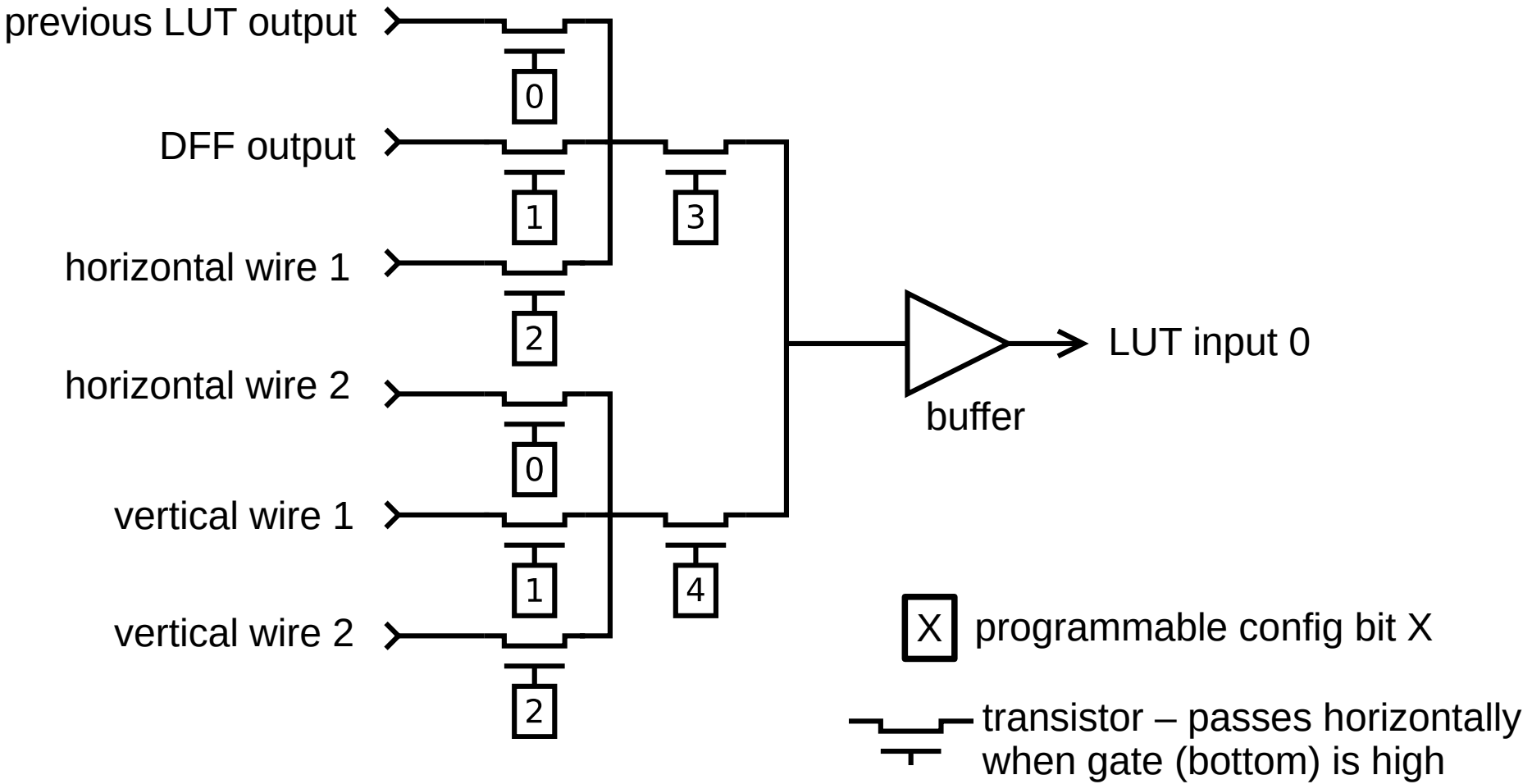
FPGA Logic

D-type flip flop (DFF) – basic storage element of an FPGA

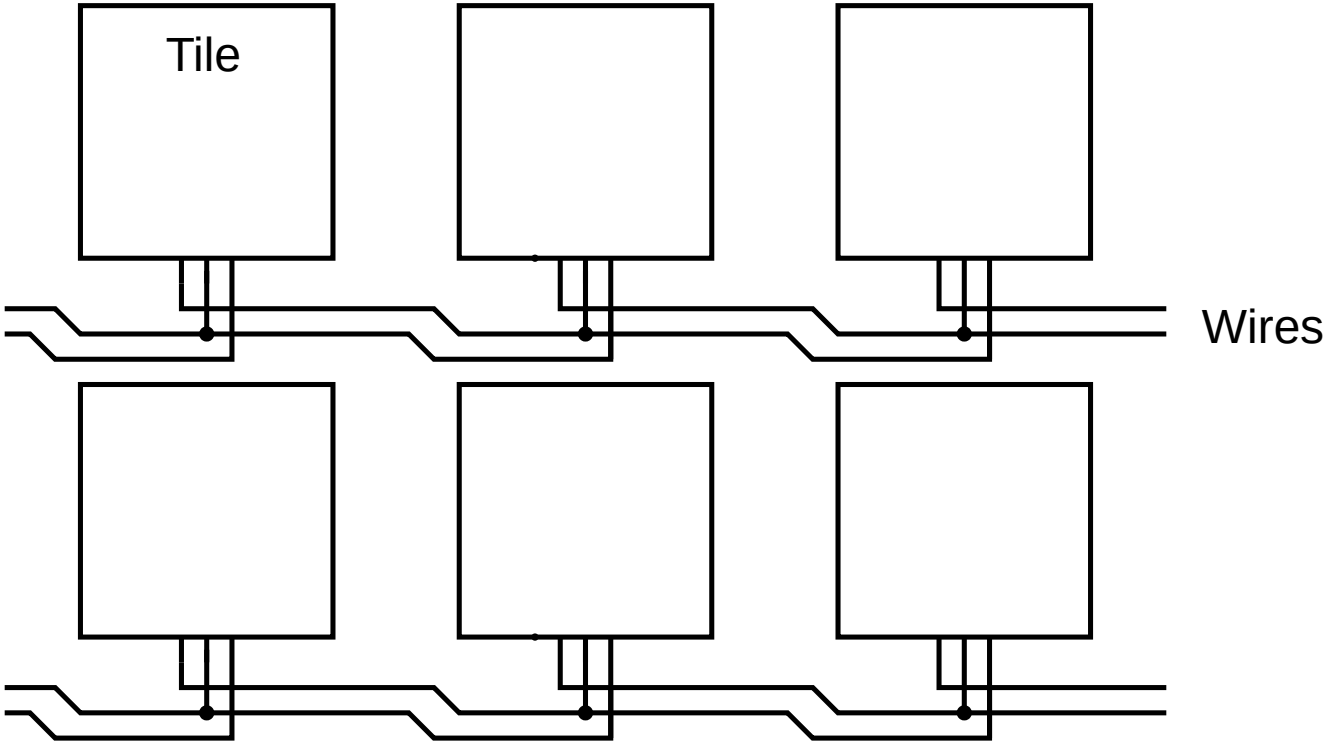
D is stored at the rising or falling edge of **CLK**

Combine DFFs and LUTs to build sequential logic such as state machines, counters, CPUs, ...

FPGA Routing

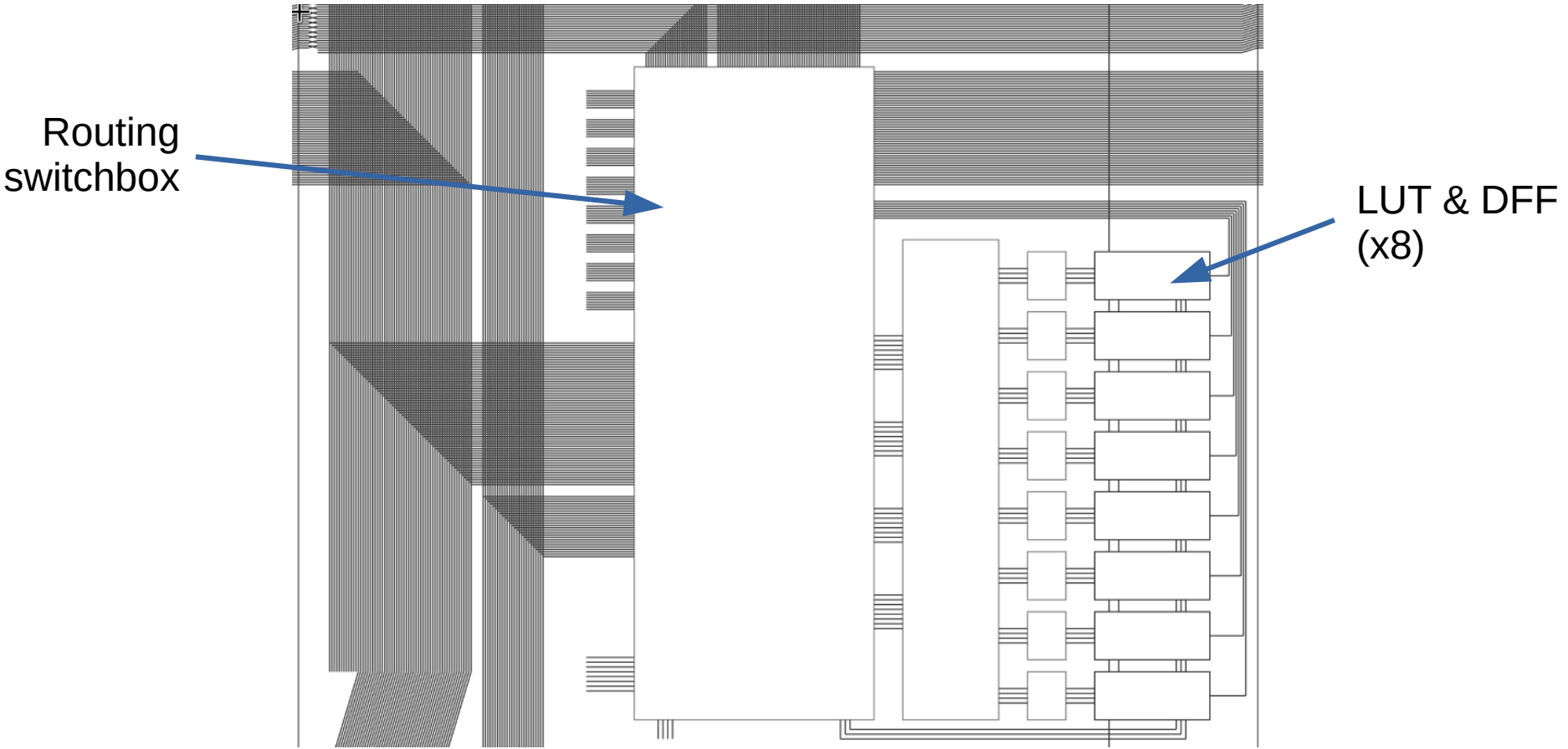


FPGA Structure



(vertical interconnect not shown for clarity)

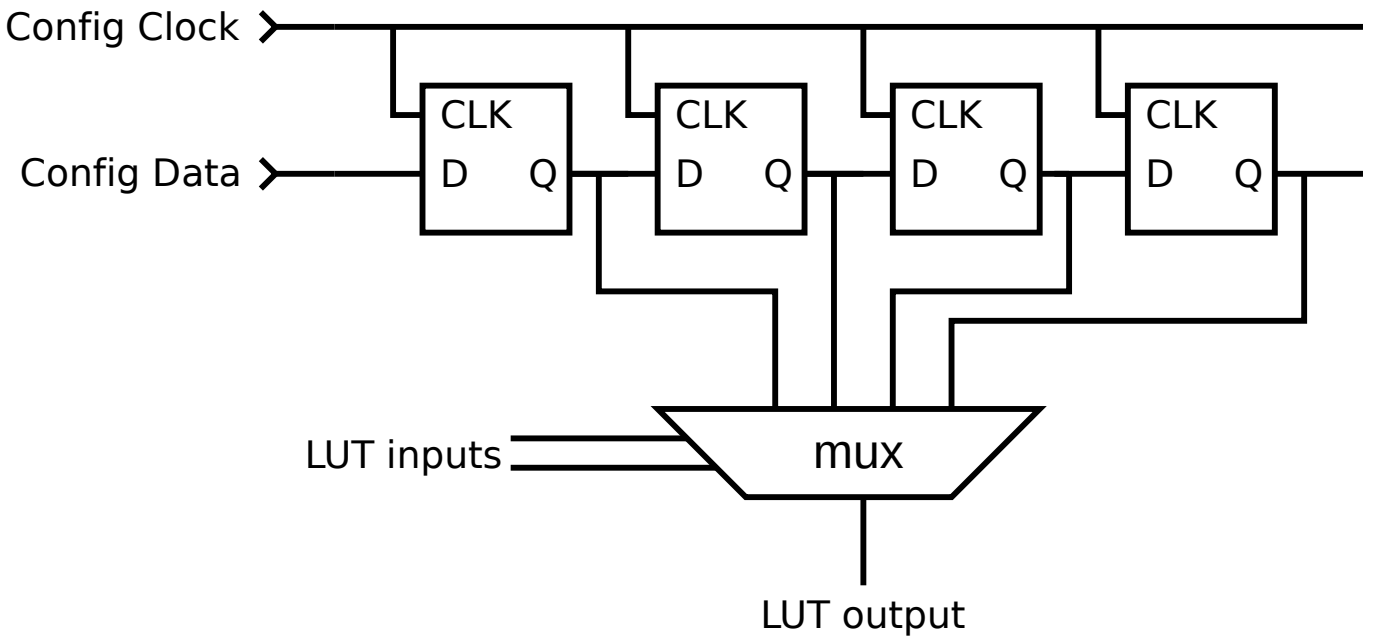
FPGA Structure



FPGA Structure

- Modern FPGAs aren’t just LUTs and DFFs!
- Block RAM: dual port SRAM, usually 4-36 kbit
- DSP: multiplier and adders
- High speed IO blocks for DDR memory, HDMI, PCI Express, etc
- Even CPUs! (Zynq ARM cores, Virtex-II Pro PPC)

FPGA Configuration



2-input FPGA LUT, showing configuration shift register

FPGA Configuration

- Logic and routing in a modern FPGA uses shift register type structures for configurability
- The programming file for an FPGA, called a **bitstream**, loads all of these shift registers
- Typical bitstreams comprised of commands, for both loading configuration and other tasks such as initialising block RAM

FPGA Configuration

- No (public) documents telling you what the bits inside a bitstream actually do!
- Unlike most other kinds of programmable chip – microcontrollers usually have register guides, etc
- Expectation is to use vendor-provided proprietary design software

Open FPGA Flows

- Need to document bitstreams to build open source tools
- **Project Icestorm**: bitstream documentation for iCE40 FPGAs by Clifford Wolf & Matthias Lasser
- Combined with Yosys & arachne-pnr to build the first useful open FPGA flow
- arachne-pnr later replaced by nextpnr

Project Trellis

- Open source bitstream docs & tools for ECP5 FPGAs
- Development started March 2018
- Basic set of bitstream docs May 2018
- Proof-of-concept flow June 2018
- Complete bitstream docs Nov 2018
- Near-complete flow Feb 2019

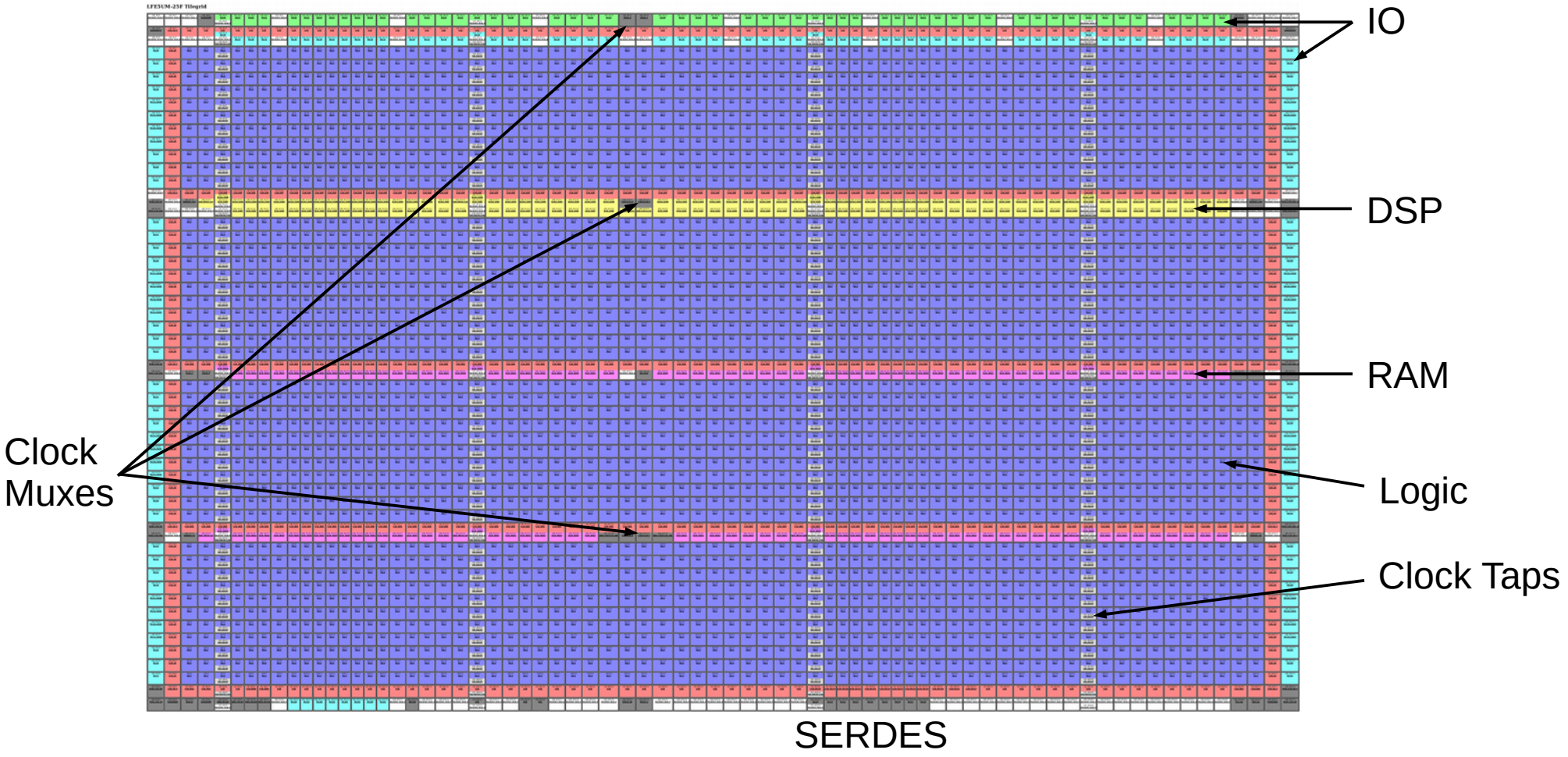
Trellis Status

- Bit and routing documentation for almost all functionality (missing: obscure DSP modes)
- Timing documentation for fabric, logic cells, IO and BRAM
- Timing-driven Yosys & nextpnr flow supporting majority of functionality including BRAM, PLL, SERDES, DDR memory IO

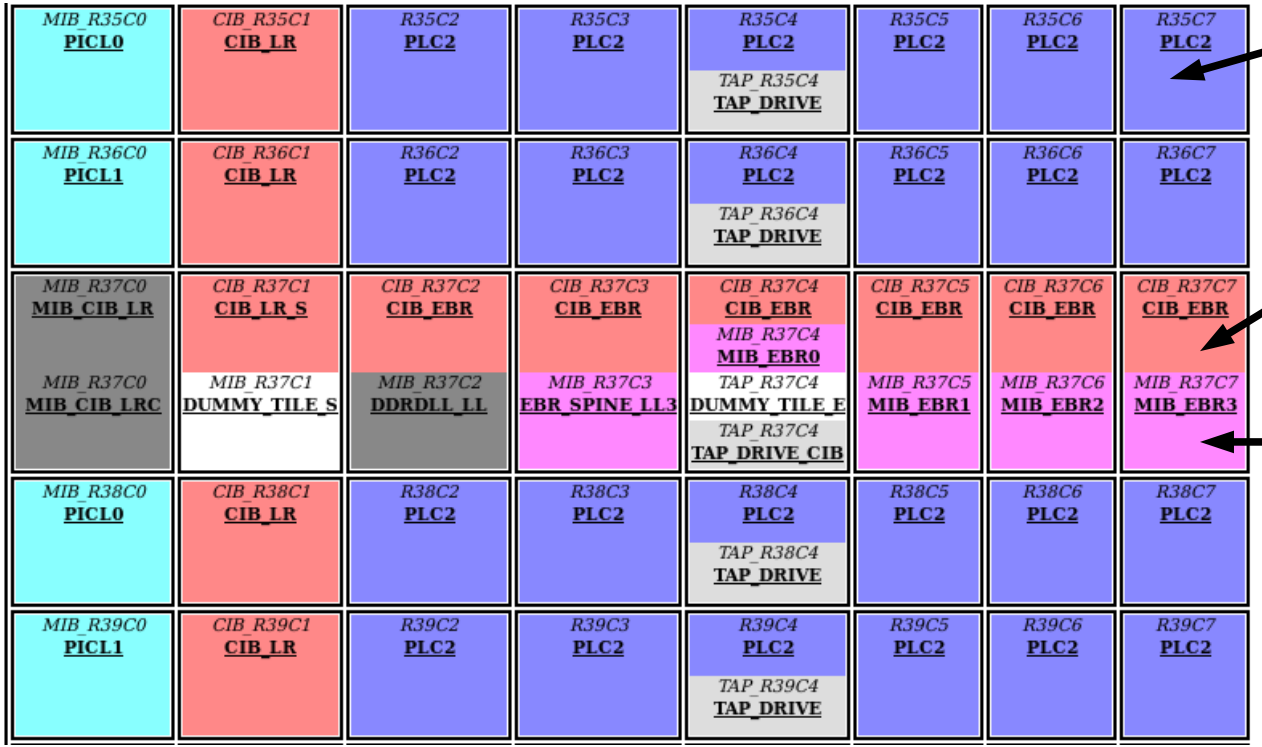
ECP5 Architecture

- Split up into **tiles** of different types. Logic tiles split into 4 **slices**
- **Slice:** 2 LUT + 2FF; carry + 2FF; 16x2 RAM + 2FF; also cascade muxes
- Fixed interconnect **wires**
- **Arcs** connect **wires** together and are configurable or fixed (aka **pip**)
- All arcs and wires are unidirectional – mux topology
- Dedicated global clock network connects to all tiles

ECP5 Architecture



ECP5 Architecture



Logic tiles contain both logic and interconnect

CIB tiles contain interconnect for non-logic functions

MIB tiles contain non-logic functionality (EBR, DSP, IO, etc)

More than one tile at a location is possible!

Low Level Bits

- First step – pack and unpack bitstreams
- ECP5 bitstreams contain various **commands**
- Chip configuration structured as **frames** of **bits**
- One command configures all frames, with a CRC after each frame’s data
- **Tiles** are a region defined by start/end frame/bit

Low Level Bits

Comment header – ignored by FPGA (ASCII strings)

Preamble

```
68 75 20 46 65 62 20 20 31 20 31 30 3A 31 35 3A 35 38
20 32 30 31 38 00 52 6F 77 73 3A 20 31 33 32 39 34 00
43 6F 6C 73 3A 20 31 31 33 36 00 42 69 74 73 3A 20 31
35 31 30 31 39 38 34 00 52 65 61 64 62 61 63 6B 3A 20
20 20 20 20 4F 66 66 00 53 65 63 75 72 69 74 79 3A 20
20 20 20 20 4F 66 66 00 42 69 74 73 74 72 65 61 6D 20
43 52 43 3A 20 30 78 36 35 45 41 00 FF FF FF BD B3 FF
FF FF FF 3B 00 00 00 E2 00 00 00 41 11 30 43 22 00 00
00 40 00 00 00 46 00 00 00 82 91 33 EE 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Dummy

Set CTRL0

Reset CRC

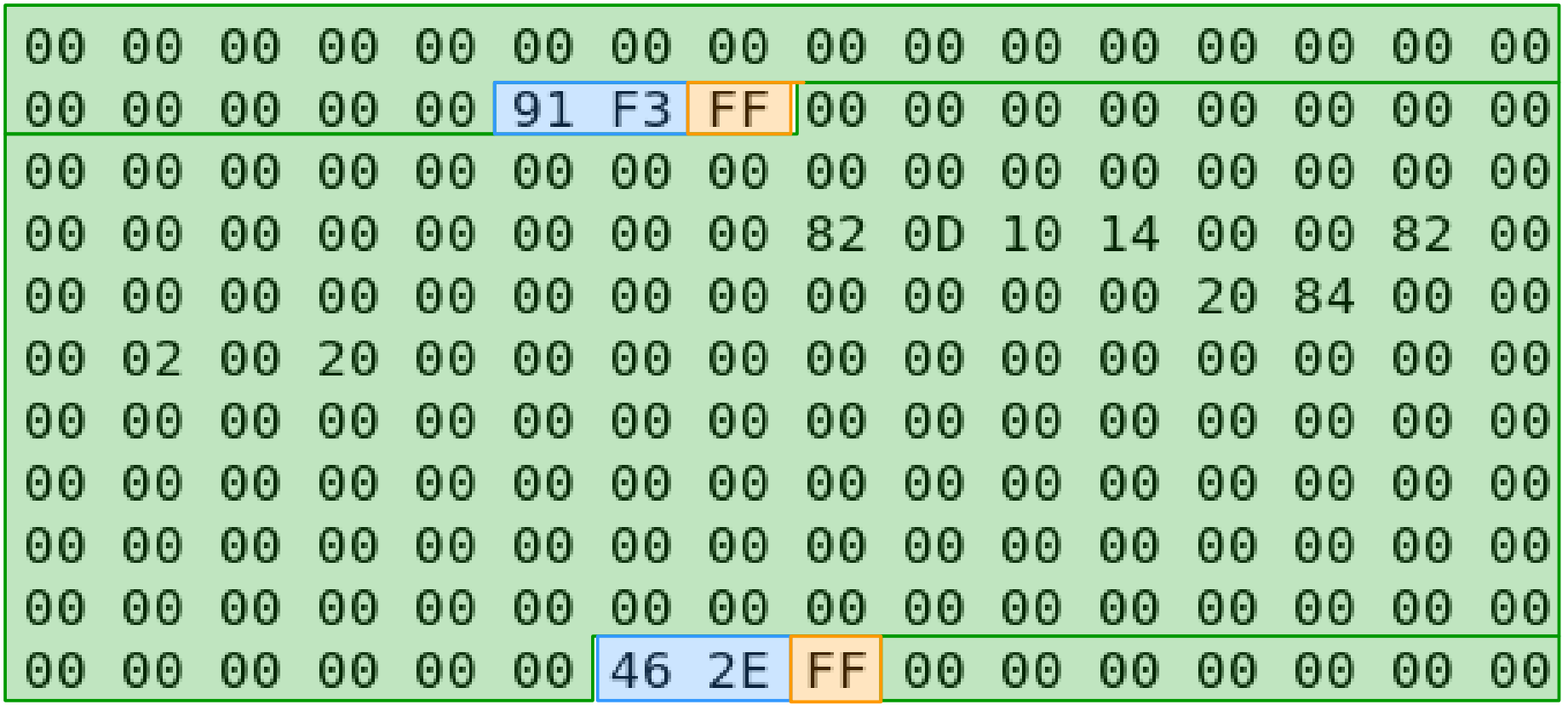
Init Address

Check IDCODE

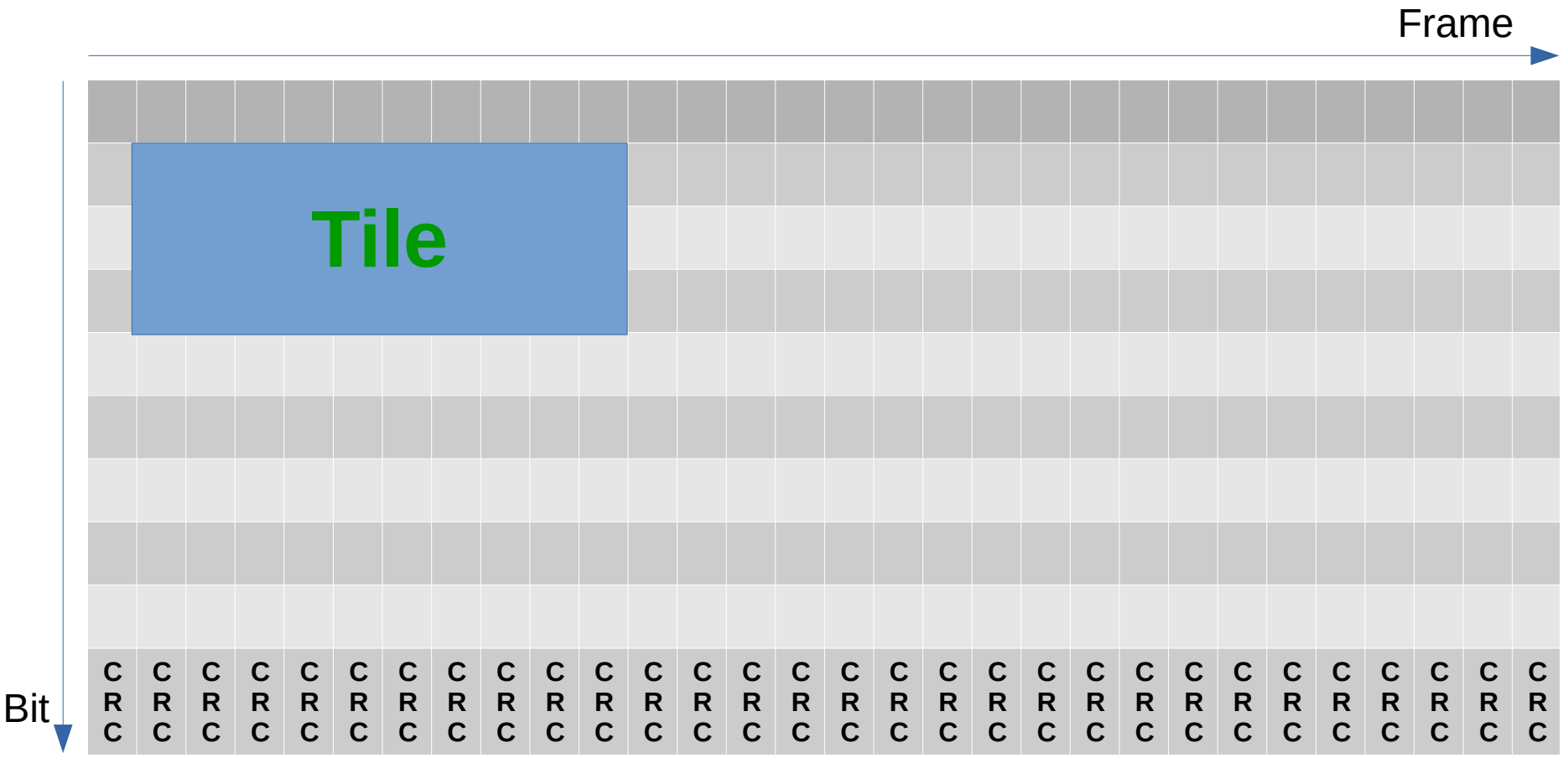
Load number frames

Frame N-1 Data

Low Level Bits



Frames and bits



Low Level Bits

- Lattice have some documentation on bitstream commands
- Trial and error needed to work out CRCs, bit ordering, etc
- Know you are good when you can round-trip to an identical bitstream

Fuzzing

- Now need to work out what the bits actually do
- Created a Python library (with Boost) to access bitstreams
- Wrote a Python framework for **fuzzers**

Fuzzing

- Use Lattice ncl files to create **post place-and-route** designs
- Faster and very targeted
- Use a Tcl API to list wires and arcs
- **Spend time looking for useful interfaces!**

Fuzzing

- Two things to fuzz: routing and non-routing config
- **Routing**
  - Use Tcl to list wires in a tile and arcs on those wires
  - Create designs for each arc with only that arc, and look at bit changes
  - Normalise net names and store in database
  - 1-2 hours for a full logic tile
  - Very automated, extends to other tiles easily

Fuzzing

- Two things to fuzz: routing and non-routing config
- **Non-routing config**
  - “Word” style configuration (LUT init): create a design with each bit set and look for changes
  - “Enum” style configuration (IO type): create a design with each possible option
  - In all cases config setting and size/possible options do need to be specified manually

Typical Fuzzer – FF modes

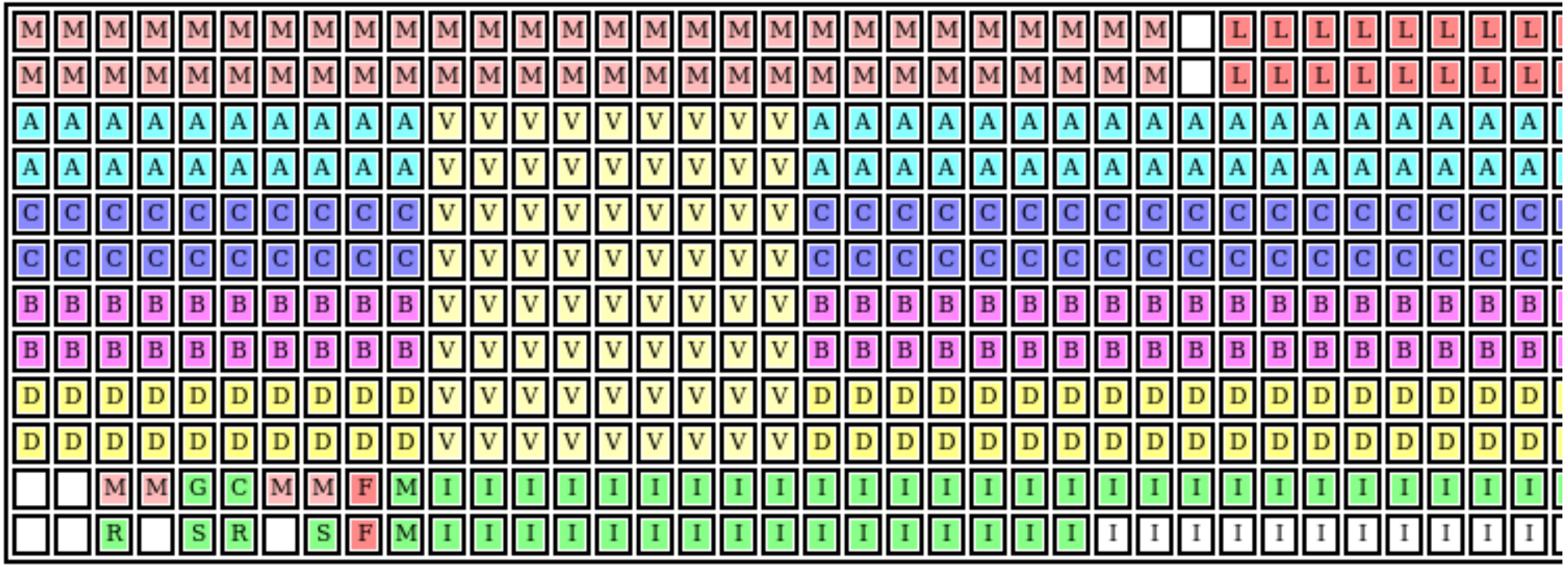
```
def get_substs(regset="RESET", sd="0", lsrmode="LSR", gsr="DISABLED"):
    return dict(slice=slicen, r=str(r), regset=regset, sd=sd, lsrmode=lsrmode, gsr=gsr)

for r in range(2):
    nonrouting.fuzz_enum_setting(cfg, "SLICE{}.REG{}.REGSET".format(slicen, r), ["RESET", "SET"],
                                 lambda x: get_substs(regset=x), empty_bitfile)
    nonrouting.fuzz_enum_setting(cfg, "SLICE{}.REG{}.SD".format(slicen, r), ["0", "1"],
                                 lambda x: get_substs(sd=x), empty_bitfile)
    nonrouting.fuzz_enum_setting(cfg, "SLICE{}.REG{}.LSRMODE".format(slicen, r), ["LSR", "PRLD"],
                                 lambda x: get_substs(lsrmode=x), empty_bitfile)
    nonrouting.fuzz_enum_setting(cfg, "SLICE{}.GSR".format(slicen), ["DISABLED", "ENABLED"],
                                 lambda x: get_substs(gsr=x), empty_bitfile)
```

Fuzzing

- Database can be rendered as HTML for manual checks

  https://symbiflow.github.io/prjtrellis-db/

Trellis Database



Database

Normalised netname
Nominal position is x+3

Frame 104, bit 9
inside tile

Mux driving E3_H06E0003

Source	F101B8	F101B9	F102B8	F103B8	F103B9	F104B8	F104B9	F105B9
W3_H06E0003	1	-	-	1	-	-	-	-
S3_V06N0003	-	1	-	1	-	-	-	-
V06N0003	-	-	1	1	-	-	-	-
W3_H06E0303	1	-	-	-	1	-	-	-
V06S0003	-	1	-	-	1	-	-	-
N3_V06S0003	-	-	1	-	1	-	-	-
W1_H02E0001	1	-	-	-	-	1	-	-
N1_V01S0000	-	1	-	-	-	1	-	-
V01N0001	-	-	1	-	-	1	-	-
W1_H02E0301	1	-	-	-	-	-	1	-
Q0	-	1	-	-	-	-	1	-
Q3	-	-	1	-	-	-	1	-
H01E0001	1	-	-	-	-	-	-	1
F0	-	1	-	-	-	-	-	1
F3	-	-	1	-	-	-	-	1

Database

Configuration word SLICEA.K0.INIT

Default value: 16'b1111111111111111

SLICEA.K0.INIT[0]	!F25B10
SLICEA.K0.INIT[1]	!F24B10
SLICEA.K0.INIT[2]	!F23B10
SLICEA.K0.INIT[3]	!F22B10
SLICEA.K0.INIT[4]	!F21B10
SLICEA.K0.INIT[5]	!F20B10
SLICEA.K0.INIT[6]	!F19B10
SLICEA.K0.INIT[7]	!F18B10
SLICEA.K0.INIT[8]	!F17B10
SLICEA.K0.INIT[9]	!F16B10
SLICEA.K0.INIT[10]	!F15B10
SLICEA.K0.INIT[11]	!F14B10
SLICEA.K0.INIT[12]	!F13B10
SLICEA.K0.INIT[13]	!F12B10
SLICEA.K0.INIT[14]	!F11B10
SLICEA.K0.INIT[15]	!F10B10

Database

Configuration Setting EBR0.DP16KD.DATA_WIDTH_A

Default value: 18

Value	F40B0	F47B0	F51B0	F78B0
1	1	1	1	1
2	1	0	1	1
4	1	0	1	0
9	0	0	1	0
18	0	0	0	0

Configuration Setting EBR0.DP16KD.WRITEMODE_A

Default value: NORMAL

Value	F7B0	F101B0
NORMAL	0	0
READBEFOREWRITE	0	1
WRITETHROUGH	1	0

Text Configuration

- Need to make use of & test fuzz results
- Tools to convert bitstreams to/from a text config format
- Check that output is logical for simple designs
- Check for unknown bits in larger designs

Text Configuration

```
.tile R53C71:PLC2
arc: A1 W1_H02E0701
arc: A3 H02E0701
arc: A4 H02E0501
arc: A5 V00B0000
arc: A7 W1_H02E0501
arc: B0 S1_V02N0301
arc: S3_V06S0303 W3_H06E0303
arc: W1_H02W0401 V02S0401
word: SLICEA.K0.INIT 1100110000000000
word: SLICEA.K1.INIT 1010101000000000
enum: SLICEA.CCU2.INJECT1_0 NO
enum: SLICEA.CCU2.INJECT1_1 NO
enum: SLICEA.D0MUX 1
enum: SLICEA.D1MUX 1
enum: SLICEA.MODE CCU2
```

Timing

- Need to know how large internal delays are to determine if a design can work at a given frequency
- Like bitstream format, not enough vendor documentation
- Delays for cells (LUTs, etc) extracted from SDF files
- Interconnect delays determined using least-squares linear fit

Timing

- Group interconnect switches into classes
- If signal is routed

  Q5 -> span2 -> span2 -> span2 -> span1 -> A5

  and has delay 1200ps
- q_sp2 + 2 * sp2_sp2 + sp2_sp1 + sp1_a = 1200

Open Source Flow

- **Yosys**: Verilog synthesis & techmapping
- **nextpnr**: place and route
- **Trellis**: bitstream generation

Yosys

- Open Verilog synthesis framework
- Support for multiple FPGA families & ASIC synthesis
- Also support for formal verification, design manipulation, ….

Yosys

- Verilog technology map rules are very flexible
- Create map rules for coarse- or fine-grained cells
- Coarse grain: adders, shift registers
- Fine grain: LUT cascade muxes, flipflops, latches
- Dedicated BRAM inference pass

Yosys

- **pmgen**: Experimental code generator framework for netlist pattern matching
- Intended for DSP inference
- Proof-of-concept iCE40 DSP inference
- ECP5 DSP inference still TODO

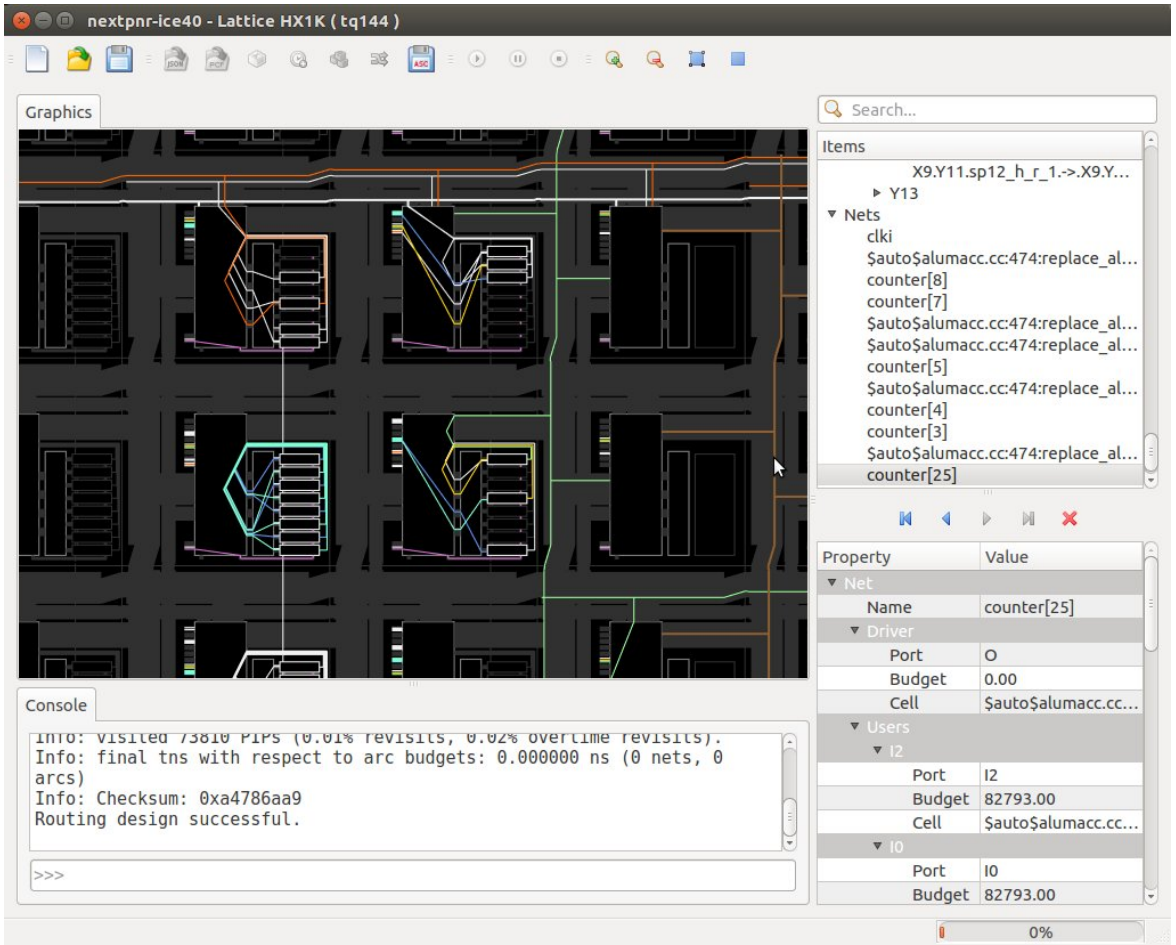
nextpnr

- New multi-architecture place-and-route
- Support for iCE40 & ECP5 FPGAs (experimental: Xilinx)
- Architectures described using code, not just flat data files
- Timing driven

nextpnr

- Analytic & SA placer options
- A*-with-ripup router
- Packer & bitstream gen provided by architecture
- Extensible using Python API

nextpnr


nextpnr-ice40 - Lattice HX1K (tq144)
Graphics
Search...
Items
X9.Y11.sp12_h_r_1.->.X9.Y...
Y13
Nets
clki
$auto$alumacc.cc:474:replace_al...
counter[8]
counter[7]
$auto$alumacc.cc:474:replace_al...
$auto$alumacc.cc:474:replace_al...
counter[5]
$auto$alumacc.cc:474:replace_al...
counter[4]
counter[3]
$auto$alumacc.cc:474:replace_al...
counter[25]
Property
Value
Net
Name
counter[25]
Driver
Port
O
Budget
0.00
Cell
$auto$alumacc.cc...
Users
I2
Port
I2
Budget
82793.00
Cell
$auto$alumacc.cc...
I0
Port
I0
Budget
82793.00
Console
Info: visited 73810 PIPs (0.01% revisits, 0.02% overtime revisits).
Info: final tns with respect to arc budgets: 0.000000 ns (0 nets, 0 arcs)
Info: Checksum: 0xa4786aa9
Routing design successful.
>>>
0%

Capabilities

- VexRiscv system-on-chip with DDR3 & 1GbE
- High-speed interfaces including HDMI
- Build & program blinky in 3 seconds
- Build & program picorv32 SoC in <20 seconds
- WIP: open PCIe PHY, ...

Capabilities