

# **Project Trellis**

## FOSS Tools for ECP5 FPGAs

David Shah  
Symbiotic EDA // Imperial College London

# Why ECP5?

- Exciting new dev boards including TinyFPGA
- Much larger than iCE40, still simple enough for a full FOSS flow
- Cheapest FPGA per LC and readily available
- Vendor tools not as good as the “big guns”

# ECP5 FPGA

- Up to 85k logic cells (LUT4+FF)
- Up to 3.7Mb BRAM (18Kb blocks), 156 18x18 DSPs
- Available with 3G or 5G SERDES

# ECP5 Architecture

- Split up into **tiles**. 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

# ECP5 Architecture

- More than one tile possible at a grid location
- Logic tiles contain both logic and interconnect
- Other functions split between “MIB” tiles containing functionality and “CIB” tiles containing interconnect
- Interconnect is identical for logic and other tiles

# ECP5 Architecture

MIB_R11C0 PICL0	CIB_R11C1 CIB_LR	R11C2 PLC2	R11C3 PLC2	TAP_R11C4 TAP_DRIVE R11C4 PLC2	R11C5 PLC2	R11C6 PLC2	R11C7 PLC2	R11C8 PLC2	R11C9 PLC2	R11C10 PLC2	R11C11 PLC2	R11C12 PLC2	R11C13 PLC2
MIB_R12C0 PICL1	CIB_R12C1 CIB_LR	R12C2 PLC2	R12C3 PLC2	TAP_R12C4 TAP_DRIVE R12C4 PLC2	R12C5 PLC2	R12C6 PLC2	R12C7 PLC2	R12C8 PLC2	R12C9 PLC2	R12C10 PLC2	R12C11 PLC2	R12C12 PLC2	R12C13 PLC2
MIB_R13C0 MIB_CIB_LR	CIB_R13C1 CIB_LR_S	CIB_R13C2 CIB_DSP	CIB_R13C3 CIB_DSP	TAP_R13C4 TAP_DRIVE_CIB CIB_R13C4 CIB_DSP	CIB_R13C5 CIB_DSP	CIB_R13C6 CIB_DSP	CIB_R13C7 CIB_DSP	CIB_R13C8 CIB_DSP	CIB_R13C9 CIB_DSP	CIB_R13C10 CIB_DSP	CIB_R13C11 CIB_DSP	CIB_R13C12 CIB_DSP	CIB_R13C13 CIB_DSP
MIB_R13C0 MIB_CIB_LRC	MIB_R13C1 DUMMY_TILE_S	MIB_R13C2 DDRDL1_ULA	MIB_R13C3 DSP_SPINE_UL1	TAP_R13C4 DUMMY_TILE_E	MIB_R13C5 MIB_DSP1	MIB_R13C6 MIB_DSP2	MIB_R13C7 MIB_DSP3	MIB_R13C8 MIB_DSP4	MIB_R13C9 MIB_DSP5	MIB_R13C10 MIB_DSP6	MIB_R13C11 MIB_DSP7	MIB_R13C12 MIB_DSP8	MIB_R13C13 MIB_DSP0
MIB_R13C0 DUMMY_TILE_7	MIB_R13C1 DUMMY_TILE_T	MIB_R13C2 DUMMY_TILE_T	MIB_R13C3 DUMMY_TILE_T	MIB_R13C4 TAP_R13C4 DUMMY_TILE_F	MIB_R13C5 MIB2_DSP1	MIB_R13C6 MIB2_DSP2	MIB_R13C7 MIB2_DSP3	MIB_R13C8 MIB2_DSP4	MIB_R13C9 MIB2_DSP5	MIB_R13C10 MIB2_DSP6	MIB_R13C11 MIB2_DSP7	MIB_R13C12 MIB2_DSP8	MIB_R13C13 MIB2_DSP0
MIB_R14C0 PICL0	CIB_R14C1 CIB_LR	R14C2 PLC2	R14C3 PLC2	TAP_R14C4 TAP_DRIVE R14C4 PLC2	R14C5 PLC2	R14C6 PLC2	R14C7 PLC2	R14C8 PLC2	R14C9 PLC2	R14C10 PLC2	R14C11 PLC2	R14C12 PLC2	R14C13 PLC2

# Current Status

- Complete bit and routing docs for logic, interconnect, BRAM and PLL tiles
- Partial docs for global network, IO and DSP tiles
- nextpnr flow supporting LUTs, FFs and IOs, can build picorv32 – proves logic docs are correct

# 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

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	91	F3	FF	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	82	0D	10	14	00	00	82	00
00	00	00	00	00	00	00	00	00	00	00	00	20	84	00	00
00	02	00	20	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	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	46	2E	FF	00	00	00	00	00	00	00

Frame data

CRC

Dummy



# 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

# Low Level Bits

- Lattice include a tool to dump set bits as text
- Somewhat similar to **glb** files used in icestorm
- Can compare our own bitstream dumps
- Even more useful: get tile offsets in terms of frames and bits

# 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

# Fuzzing

- Database can be rendered as HTML for manual checks

**<https://symbiflow.github.io/prjtrellis-db/>**



# Database

Normalised netname

Nominal position is x+3

Mux driving **E3\_H06E0003**

Frame 104, bit 9  
inside tile

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
```



# Place and Route

- Already working on nextpnr multi-platform FOSS PnR, starting with iCE40 arch
- More on that in Clifford's presentation next
- Next step: adding ECP5 arch to nextpnr

# Place and Route

- iCE40 architecture uses a flat database, containing details of everything in the chip with no attempt to remove repetition
- ECP5 much larger, this would mean a database in the GB
- So we use a deduplicated database approach

# Place and Route

- During database creation, we build a full flat database in memory using relative coordinates
- Then we split it into grid locations, and use a hash to find identical grid locations
- Identical grid locations only need their content to be stored once in the final database

# Place and Route

- Text configuration used as an intermediate format between nextpnr and bitstream generation
- Avoids need for low level bitstream code in nextpnr

# Next Steps – Help Wanted

- Adding PnR support for block RAM, distributed RAM and carries
- Finish DSP documentation and add synthesis and PnR support (extensions for inference)
- Creating a PLL config tool and adding PLLs to PnR
- Working on JTAG/programming tools
- Documenting advanced IO features and IO
- Improve timing documentation and create timing tool