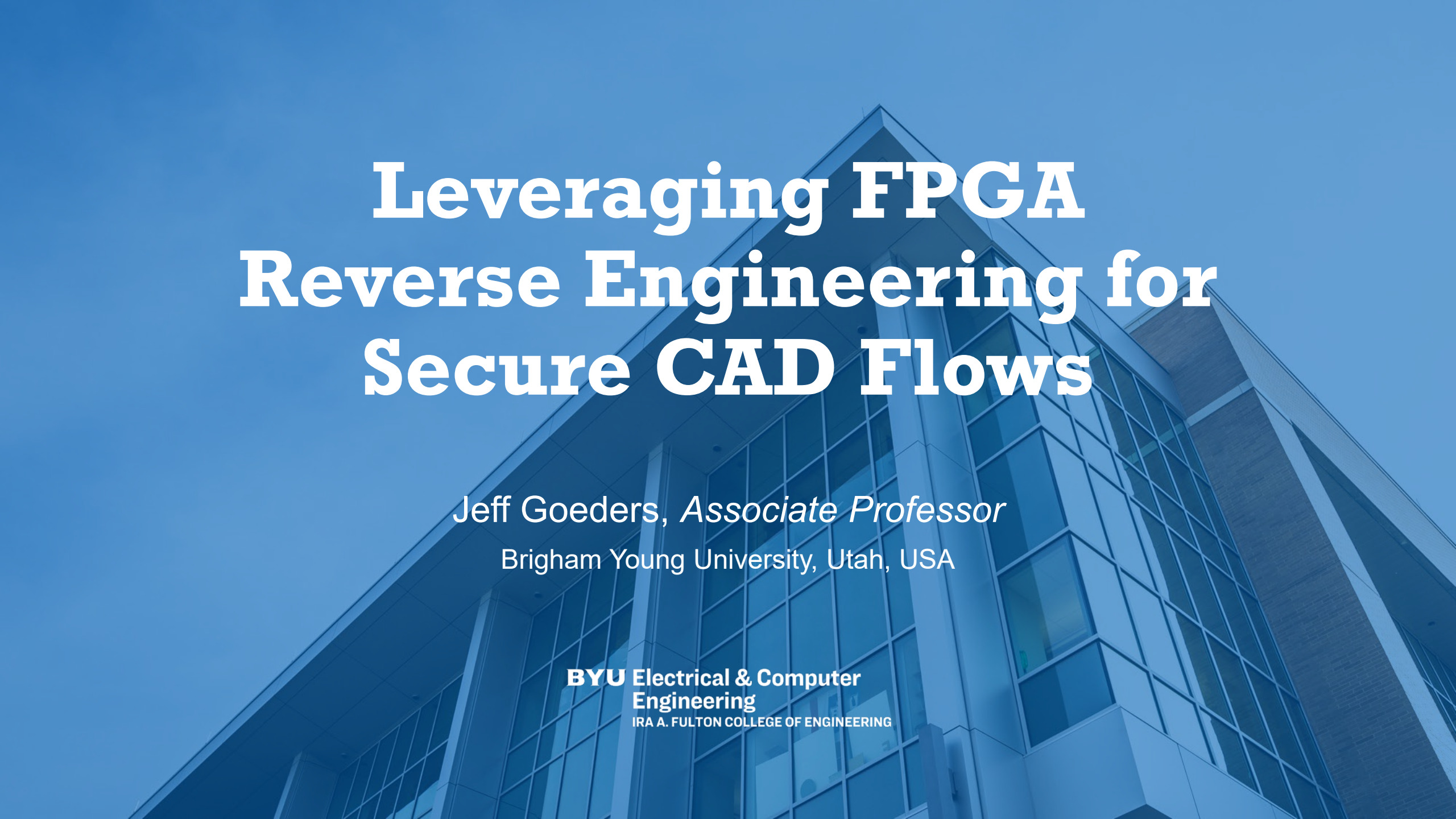


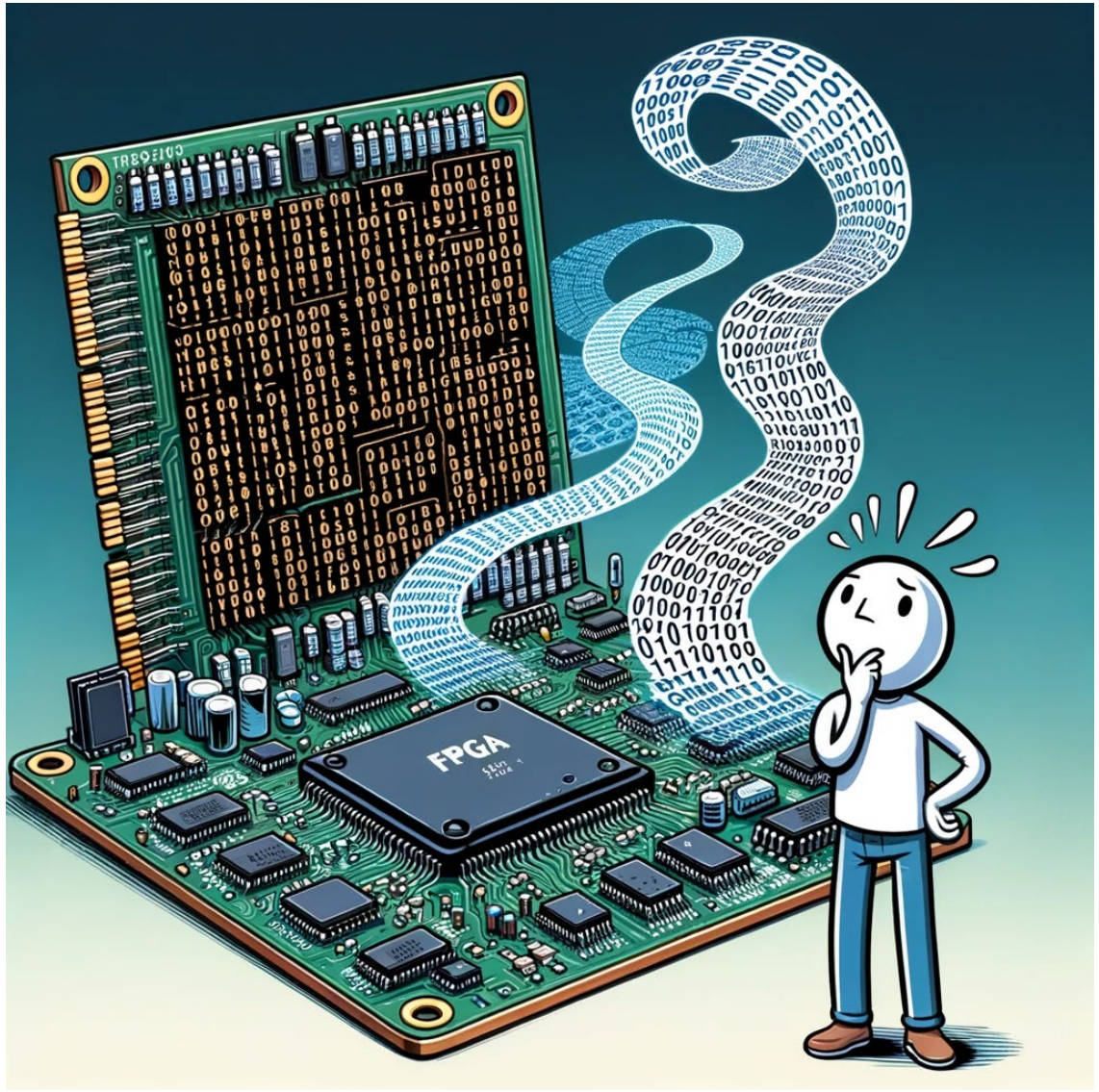
Leveraging FPGA Reverse Engineering for Secure CAD Flows

Jeff Goeders, *Associate Professor*

Brigham Young University, Utah, USA

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

FPGA Bitstreams



FPGA vendors keep bitstream formats **proprietary and secre**t.

This has historically provided some protection against certain attacks (eg IP theft)

Recent open-source work has **reverse-engineered several bitstream formats**.

This work: How can we leverage these known bitstream formats to **enhance security of FPGA design flows**?

Project 1: Verifying bitstream-to-netlist equivalence

Reilly McKendrick, Keenan Faulkner, and Jeffrey Goeders, “Assuring Netlist-to-Bitstream Equivalence using Physical Netlist Generation and Structural Comparison,” International Conference on Field-Programmable Technology (FPT), Dec 2023.

Project 2: Protecting Encrypted IP

Daniel Hutchings, Adam Taylor, Jeffrey Goeders, “Toward Intellectual Property (IP) Encryption from Netlist to Bitstream”, ACM Transactions on Reconfigurable Technology and Systems (TRETS), *to be published*, 2024.

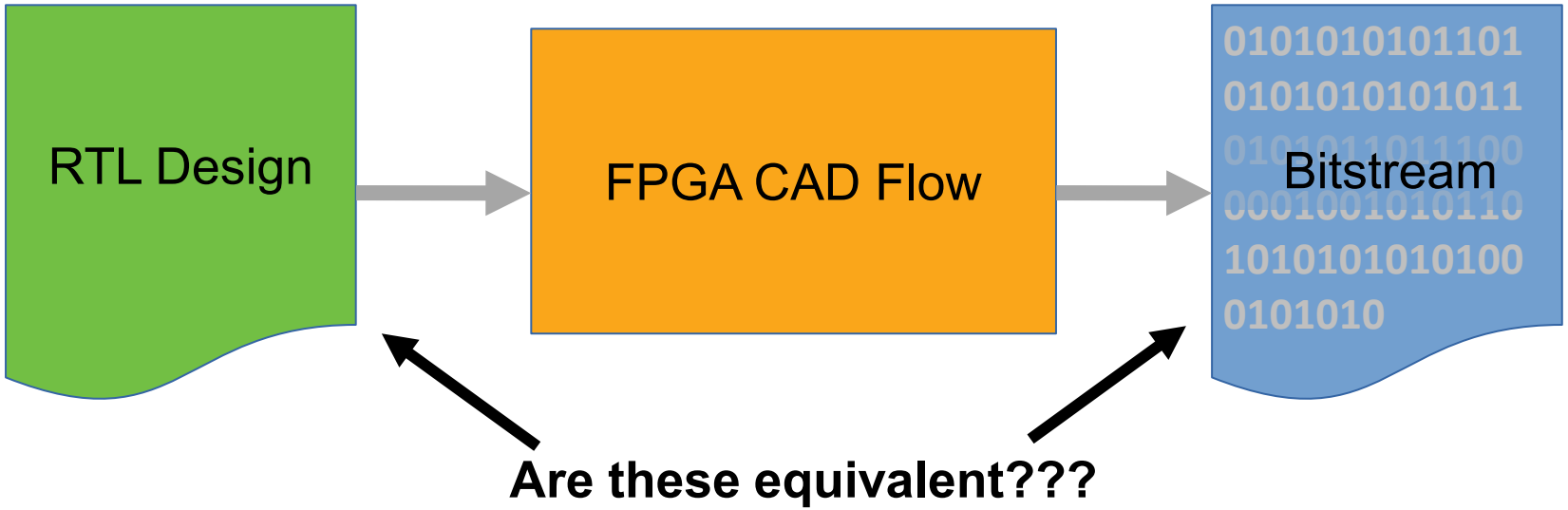
Project 1: Verifying bitstream-to-netlist equivalence

Reilly McKendrick, Keenan Faulkner, and Jeffrey Goeders, “Assuring Netlist-to-Bitstream Equivalence using Physical Netlist Generation and Structural Comparison,” International Conference on Field-Programmable Technology, Dec 2023.

Project 2: Protecting Encrypted IP

Daniel Hutchings, Adam Taylor, Jeffrey Goeders, “Toward Intellectual Property (IP) Encryption from Netlist to Bitstream”, ACM Transactions on Reconfigurable Technology and Systems, *to be published*, 2024.

Big Picture Goal



Currently, there is no easy way for a designer to figure this out…

Why should someone care?

Threat Model: Design is modified maliciously (or accidentally) during compilation, bitgen, or post-bitgen.

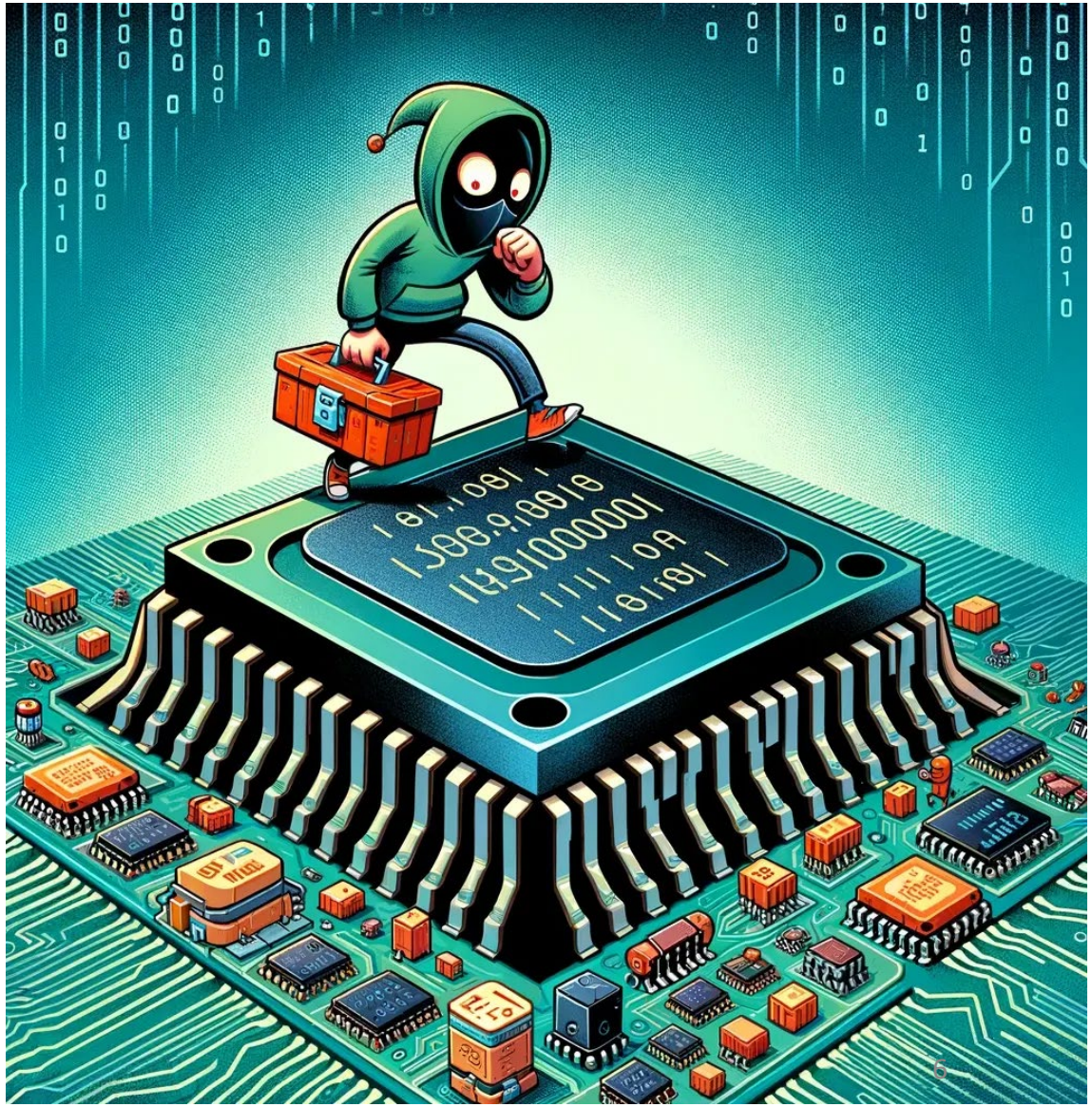
Possible attack scenarios:

1. Malicious CAD tool.
2. Buggy CAD tools accidentally modify design.
3. Attacker intercepts and modifies bitstream post-generation.

Possibility of these types of attacks have been demonstrated:

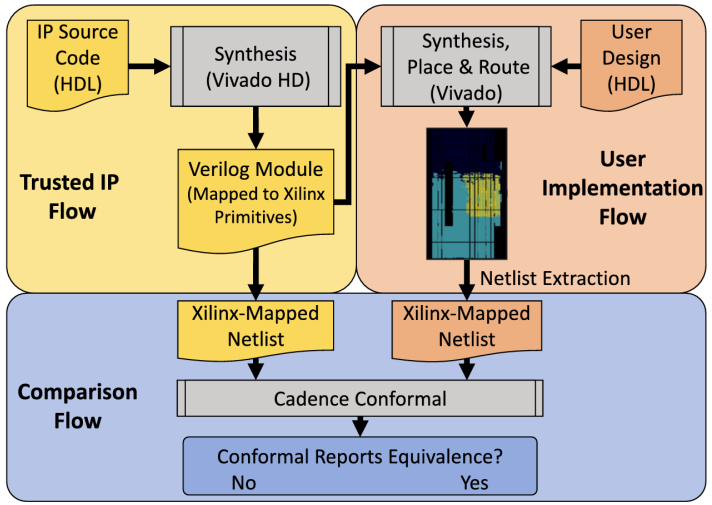
[1] R. S. Chakraborty, I. Saha, A. Palchaudhuri, and G. K. Naik, “Hardware Trojan Insertion by Direct Modification of FPGA Configuration Bitstream,” IEEE Design Test, vol. 30, no. 2, pp. 45–54, Apr. 2013, doi: 10.1109/MDT.2013.2247460.

[2] C. Krieg, C. Wolf, and A. Jantsch, “Malicious LUT: A stealthy FPGA trojan injected and triggered by the design flow,” in International Conference on Computer-Aided Design (ICCAD), Nov. 2016, pp. 1–8.



Our Previous Work

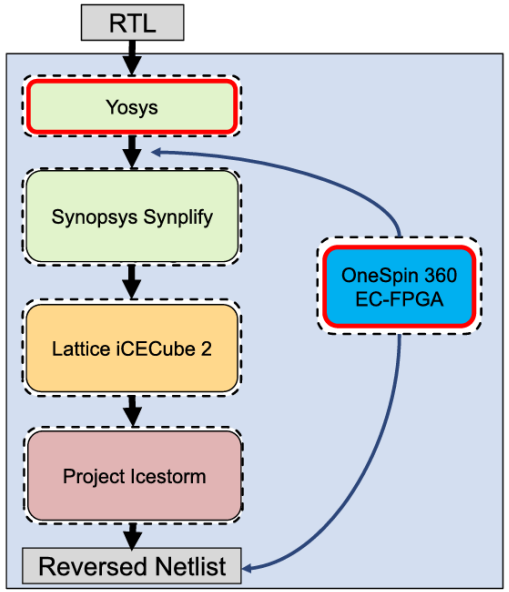
Approach #1: Validate a single IP using netlist information from Vivado and a commercial formal equivalence checking tool.



Hastings, S. Jensen, J. Goeders, and B. Hutchings, "Using physical and functional comparisons to assure 3rd-party IP for modern FPGAs," in International Verification and Security Workshop (IVSW), Jul. 2018, pp. 80–86.

Issue: Trusts CAD tool to correctly report design information

Approach #2: Use bitstream-to-netlist tool (Project Icestorm) plus a commercial formal equivalence checker.

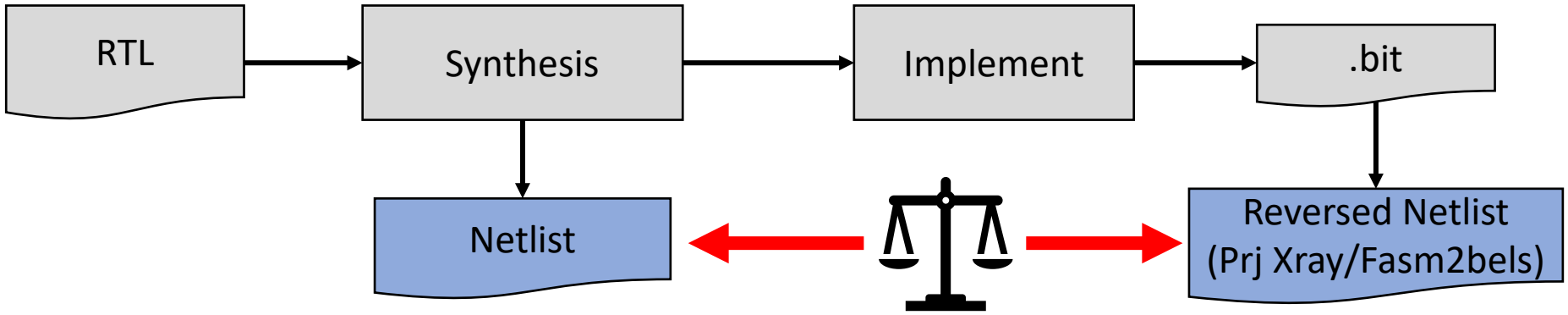


H. Cook, J. Arscott, B. George, T. Gaskin, J. Goeders, and B. Hutchings, "Inducing non-uniform FPGA aging using configuration-based short circuits," ACM Transactions on Reconfigurable Technology and Systems, vol. 15, no. 4, 41:1–41:33, Jun. 6, 2022.

Issue: Scalability is challenging. Formal verification tools can have very long run times. (hours for ~4000 LUT design)

This work: Scalable bitstream-to-netlist equivalence checking. Doesn't trust the CAD tools.

Equivalence Flow



Bitstream Equivalence Checking

Post-Synthesis Netlist

```
LUT6_2 #(
  .INIT(64'h0CFA0CFA0C0A0C0A))
  \result_OBUF[14]_inst_i_21_X55Y117_B6LUT_phys
    (.I0(op1_IBUF[14]),
     .I1(op1_IBUF[22]),
     .I2(op2_IBUF[4]),
     .I3(op2_IBUF[3]),
     .I4(\<const1> ),
     .I5(op1_IBUF[30]),
     .O6(\result_OBUF[14]_inst_i_21_n_0 ));
LUT6_2 #(
  .INIT(64'hCCCCF0F0AAAAFF00))
  \result_OBUF[14]_inst_i_17_X55Y117_A6LUT_phys
    (.I0(\result_OBUF[14]_inst_i_19_n_0 ),
     .I1(\result_OBUF[14]_inst_i_22_n_0 ),
     .I2(\result_OBUF[14]_inst_i_20_n_0 ),
     .I3(\result_OBUF[14]_inst_i_21_n_0 ),
     .I4(op2_IBUF[1]),
     .I5(op2_IBUF[2]),
     .O6(\result_OBUF[14]_inst_i_17_n_0 ));
```

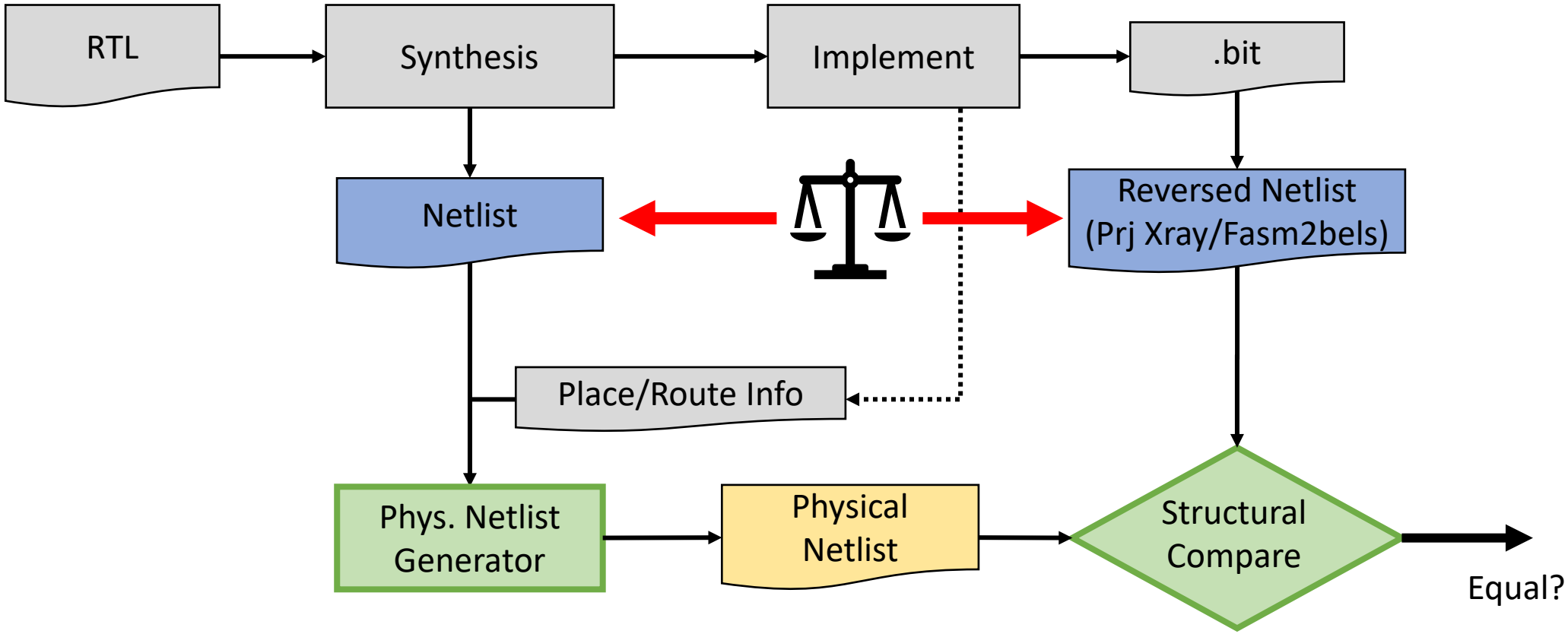
Post-Bitstream Netlist

```
9020    (* KEEP, DONT_TOUCH, BEL = "B6LUT" *)
9021    LUT6_2 #(
9022      .INIT(64'h0cfa0cfa0c0a0c0a)
9023    ) CLBLL_L_X36Y117_SLICE_X55Y117_BLUT (
9024  .I0(RIOB33_SING_X105Y100_IOB_X1Y100_I),
9025  .I1(RIOB33_X105Y107_IOB_X1Y108_I),
9026  .I2(LIOB33_X0Y117_IOB_X0Y118_I),
9027  .I3(LIOB33_X0Y117_IOB_X0Y117_I),
9028  .I4(1'b1),
9029  .I5(RIOB33_X105Y115_IOB_X1Y116_I),
9030  .O5(CLBLL_L_X36Y117_SLICE_X55Y117_BO5),
9031  .O6(CLBLL_L_X36Y117_SLICE_X55Y117_BO6)
9032    );
9033
9034
9035    (* KEEP, DONT_TOUCH, BEL = "A6LUT" *)
9036    LUT6_2 #(
9037      .INIT(64'hccccf0f0aaaaff00)
9038    ) CLBLL_L_X36Y117_SLICE_X55Y117_ALUT (
9039  .I0(CLBLM_R_X37Y116_SLICE_X56Y116_AO5),
9040  .I1(CLBLL_L_X36Y116_SLICE_X55Y116_BO6),
9041  .I2(CLBLL_L_X36Y116_SLICE_X55Y116_AO6),
9042  .I3(CLBLL_L_X36Y117_SLICE_X55Y117_BO6),
9043  .I4(LIOB33_X0Y115_IOB_X0Y115_I),
9044  .I5(LIOB33_X0Y115_IOB_X0Y116_I),
9045  .O5(CLBLL_L_X36Y117_SLICE_X55Y117_AO5),
9046  .O6(CLBLL_L_X36Y117_SLICE_X55Y117_AO6)
9047    );
```

Why is this hard?

- No signal names
- No instance names
- Inputs reordered
- INIT changed
- Primitives changed and combined.
- Routing resources

Equivalence Flow



Key Approach:

1. Extract from Vivado the **set of all transformations** performed on the design during Implementation.
2. Apply these transformations to the netlist, producing a **PHYSICAL NETLIST**.
3. Perform a **structural equivalence** comparison between physical netlist and reversed.

LUT Pin Reording

Post-Synthesis LUT

```
LUT6 #(
  .INIT(64'hEEEEEEE0EEE0EEE0))
  \result_OBUF[27]_inst_i_1
    (.I0(\result_OBUF[27]_inst_i_2_n_0 ),
     .I1(\result_OBUF[27]_inst_i_3_n_0 ),
     .I2(\result_OBUF[27]_inst_i_4_n_0 ),
     .I3(\result_OBUF[27]_inst_i_5_n_0 ),
     .I4(\result_OBUF[27]_inst_i_6_n_0 ),
     .I5(\result_OBUF[27]_inst_i_7_n_0 ),
     .O(result_OBUF[27]));
```

Post-Implementation LUT

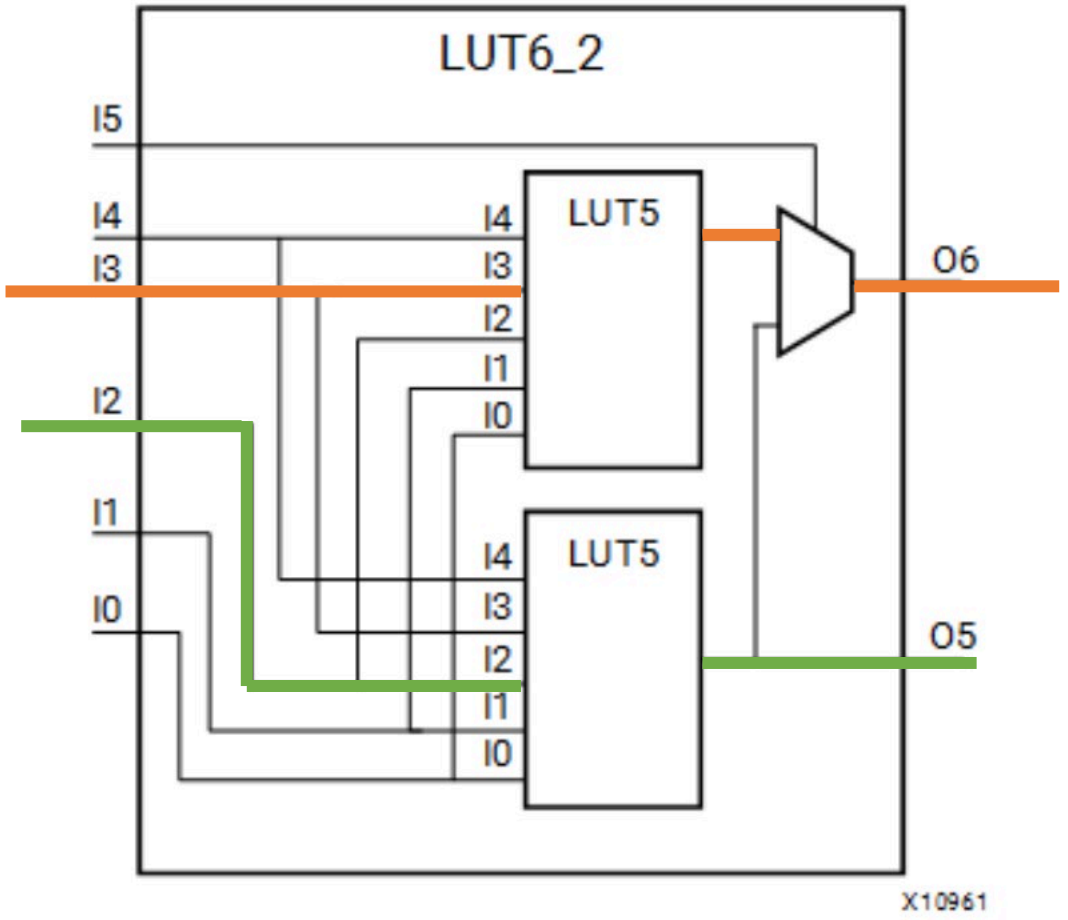
```
LUT6_2 #(
  .INIT(64'hFFF0FFF0EEE0CCC0))
  \result_OBUF[27]_inst_i_1_X57Y127_A6LUT_phys
    (.I0(\result_OBUF[27]_inst_i_7_n_0 ),
     .I1(\result_OBUF[27]_inst_i_4_n_0 ),
     .I2(\result_OBUF[27]_inst_i_2_n_0 ),
     .I3(\result_OBUF[27]_inst_i_3_n_0 ),
     .I4(\result_OBUF[27]_inst_i_6_n_0 ),
     .I5(\result_OBUF[27]_inst_i_5_n_0 ),
     .O6(result_OBUF[27]));
```

Example: LUT Routethru

Post-Synthesis Netlist

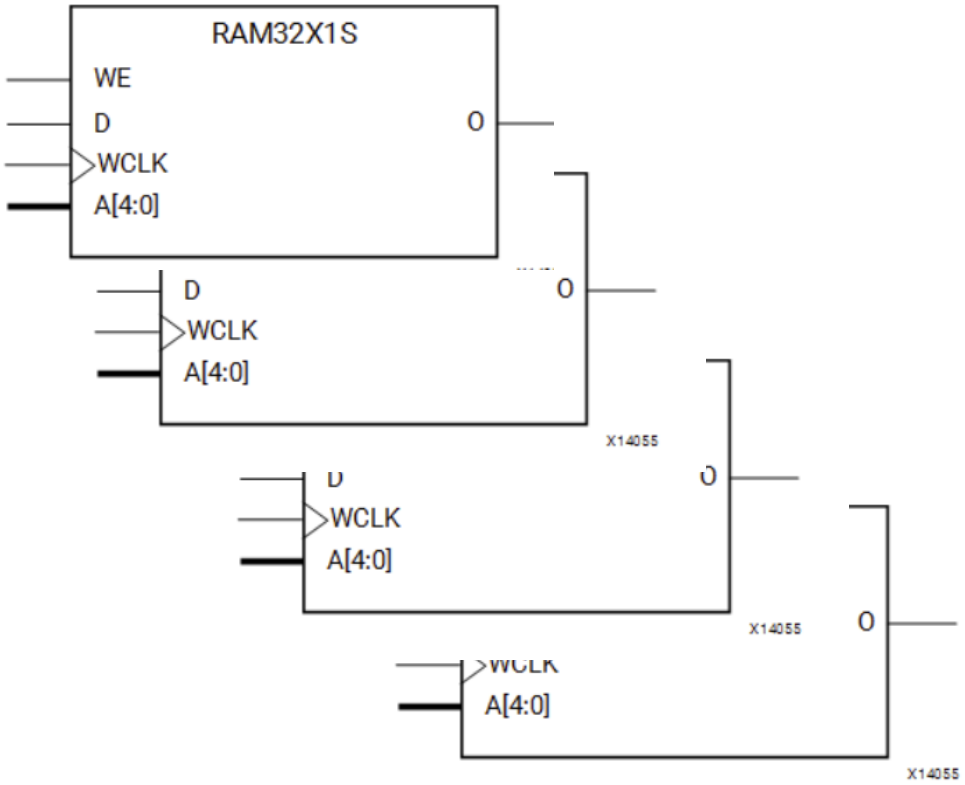
Nothing there!

Post-Bitstream Netlist

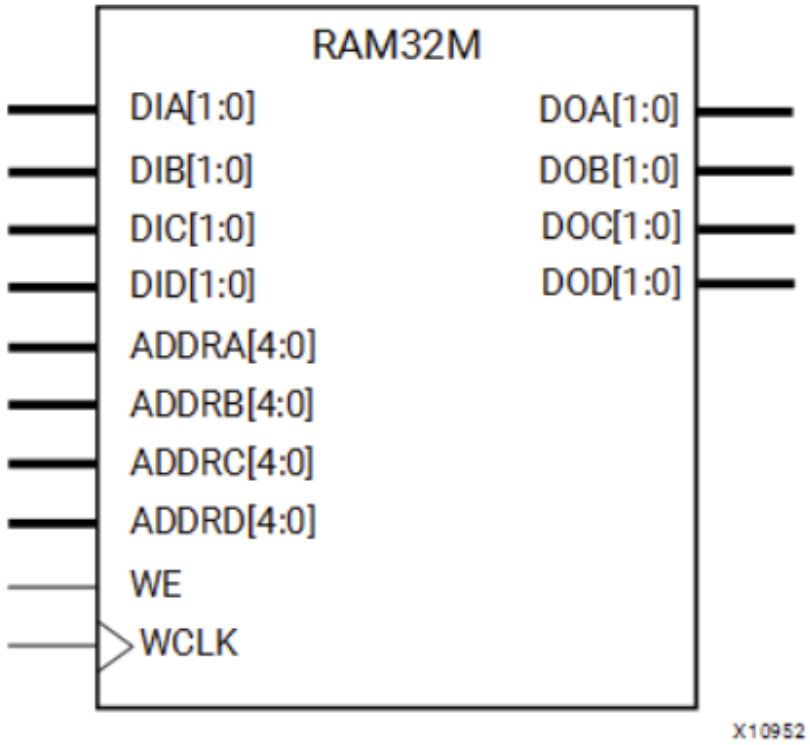


LUTRAMs

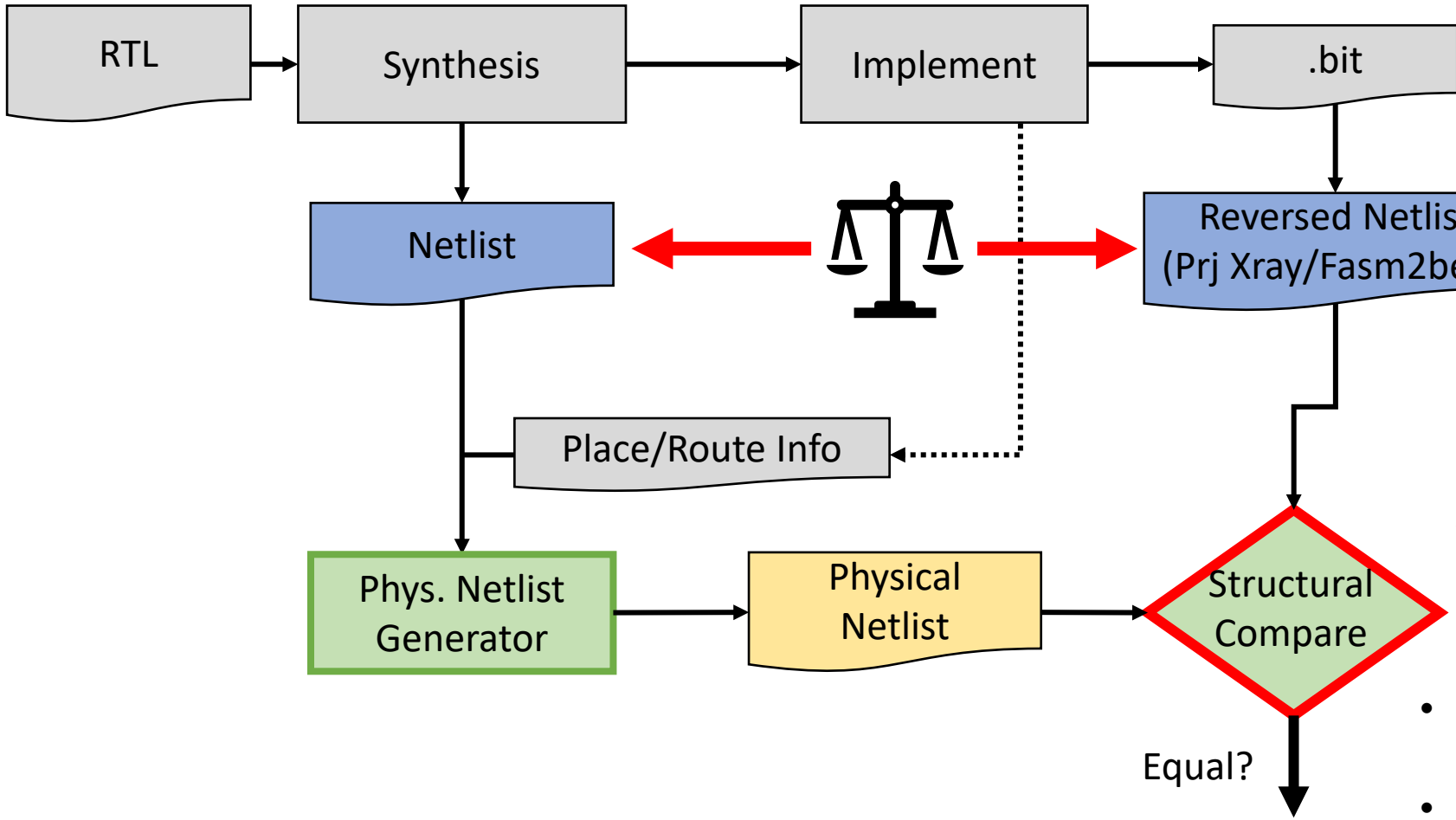
Post-Synthesis Netlist

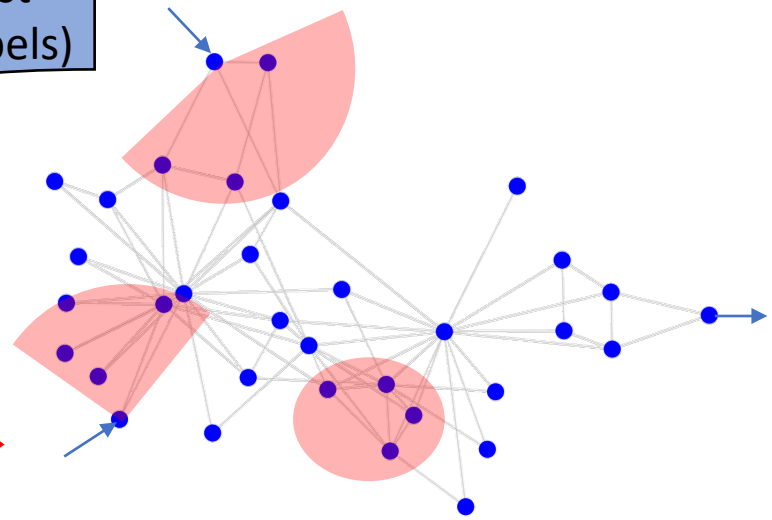


Post-Bitstream Netlist



Structural Comparison





- Structural graph matching is much easier than formal logic equivalence
- Start at I/Os and other unique nodes and expand until entire graph matched

Result: Fast equivalence checking between FPGA post-synthesis netlist and bitstream

Results:

Runtime + Validation

TABLE I: Validated Designs

Design	Resources						Runtime (s)		# Error Injection Runs
	# LUTs	# FF	# CARRY4	# BRAM	# LUTRAM	# SRL	Phys. Netlist.	Struct. Cmp.	
stereovision1	13164	11588	2014	0	0	0	8.7	237.4	100
aes128	2790	4480	0	86	0	0	3.6	42.8	100
riscv_final	1499	1390	44	0	0	0	4.8	7.7	100
cpu8080	1010	243	86	0	0	0	1.8	2.7	100
sha	1000	894	56	0	0	0	1.5	3.4	100
mkSMAdapter4B	987	1126	73	4	0	0	1.1	4.8	100
bubblesort	814	1782	0	0	0	1	1.5	5.7	100
pid	741	423	0	0	0	0	1.2	3.3	100
median	740	125	52	0	0	0	1.2	3.1	100
a25_decode	677	640	0	0	0	0	0.7	5.0	100
regfile	611	1056	0	0	0	0	3.7	5.6	100
riscvSimpleDatapath	570	63	28	0	0	0	3.7	2.5	100
basicrsa	540	459	72	0	0	0	0.7	1.9	100
hight	502	134	28	0	0	0	0.7	1.3	100
alu	461	0	20	0	0	0	3.3	2.6	100
a25_wishbone	422	818	0	0	0	0	0.6	2.7	100
uart2spi	369	410	6	0	0	0	0.9	1.2	100
quadratic_func	238	118	52	0	0	0	0.4	1.4	100
raygentop	221	303	4	0	0	1	0.5	1.4	100
pci_mini	219	333	0	0	0	0	0.6	1.9	100
tiny_encryption_algorithm	200	264	40	0	0	0	0.5	2.4	100
data_path	179	257	3	0	0	0	0.5	1.6	100
EX_stage	168	38	4	0	0	0	0.4	0.9	100
calc	163	18	12	0	0	0	3.1	1.3	100
pic	133	77	8	0	0	0	0.4	0.4	100
wb_lcd	87	80	5	0	0	0	0.3	0.5	100
control_unit	78	5	0	0	0	0	0.3	0.4	100
a25_coprocessor	74	171	0	0	0	0	0.3	0.6	100
uart	69	137	18	0	0	0	3.0	1.5	100
stereovision3	54	118	0	0	0	0	0.1	0.4	100
shiftReg	51	20	0	0	0	0	3.0	1.1	100
UpDownButtonCount	49	24	12	0	0	0	3.0	1.3	100
simon_core	35	27	0	0	0	12	0.2	0.3	100
stopwatch	34	52	10	0	0	0	2.9	1.1	100
stereovision2	28	39	0	0	0	0	0.1	0.3	100
ID_stage	26	73	0	0	0	0	0.2	0.5	100
bcd_adder	24	50	5	0	0	0	0.3	0.2	100
uart_rx	20	39	4	0	0	0	0.1	0.2	100
rx	19	39	4	0	0	0	2.8	1.3	100
random_pulse_generator	4	33	0	0	0	0	0.2	0.1	100
a25_write_back	1	44	0	0	0	0	0.1	0.4	100
MEM_stage	0	37	0	0	64	0	0.2	0.3	100

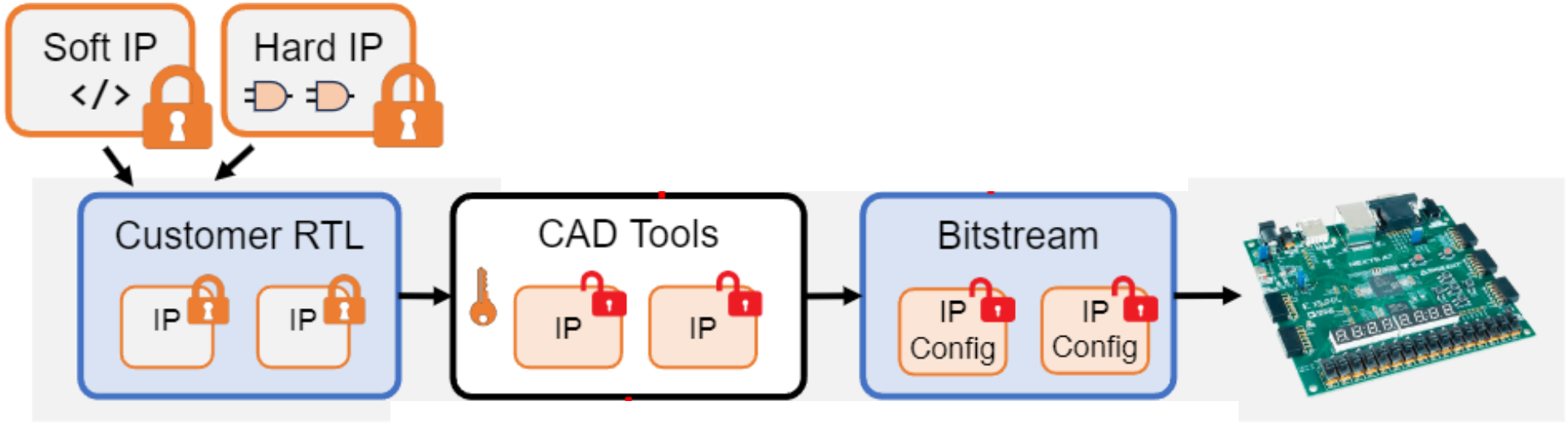
Project 1: Verifying bitstream-to-netlist equivalence

Reilly McKendrick, Keenan Faulkner, and Jeffrey Goeders, “Assuring Netlist-to-Bitstream Equivalence using Physical Netlist Generation and Structural Comparison,” International Conference on Field-Programmable Technology, Dec 2023.

Project 2: Protecting Encrypted IP

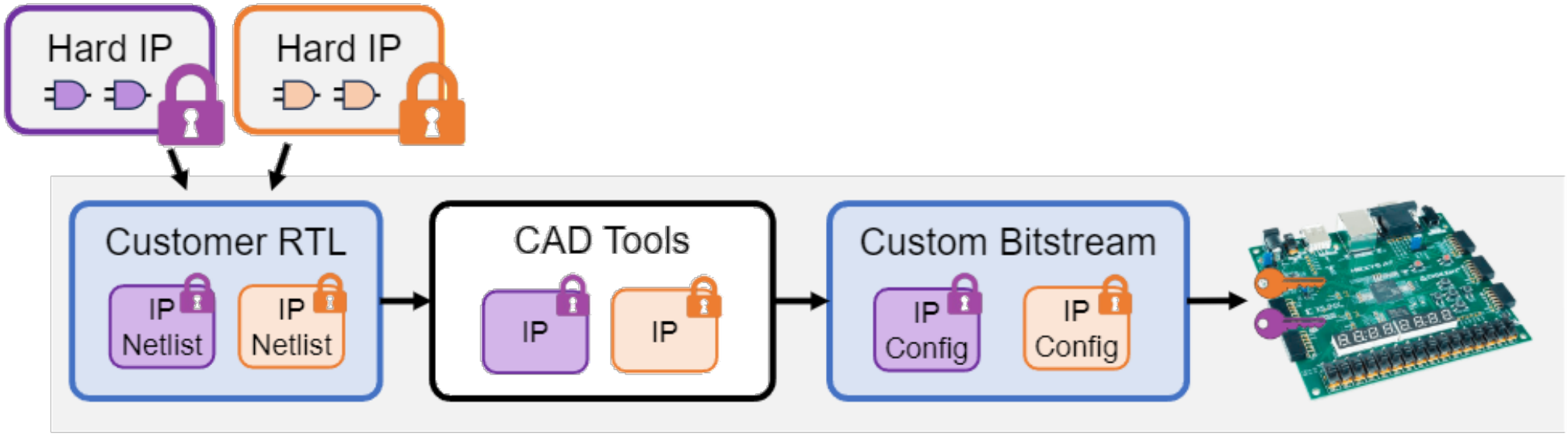
Daniel Hutchings, Adam Taylor, Jeffrey Goeders, “Toward Intellectual Property (IP) Encryption from Netlist to Bitstream”, ACM Transactions on Reconfigurable Technology and Systems, *to be published*, 2024.

Encrypted IP



FPGA “bitstream encryption” doesn’t solve this problem. It stops in-field capture of the bitstream.
It doesn’t prevent the IP customer from viewing the IP.

Ideal: IP Encryption Framework



Problems:

1. Requires new CAD tools
   - How can you do CAD on encrypted IP?

2. Requires new FPGA devices
   - Perform fine-grained decryption during configuration

Proof-of-Concept Tool

Our goal: Demonstrate an end-to-end encryption of third-party IP

Use an **existing commercial CAD tool**

Use an **existing commercial FPGA**

Is this even possible?

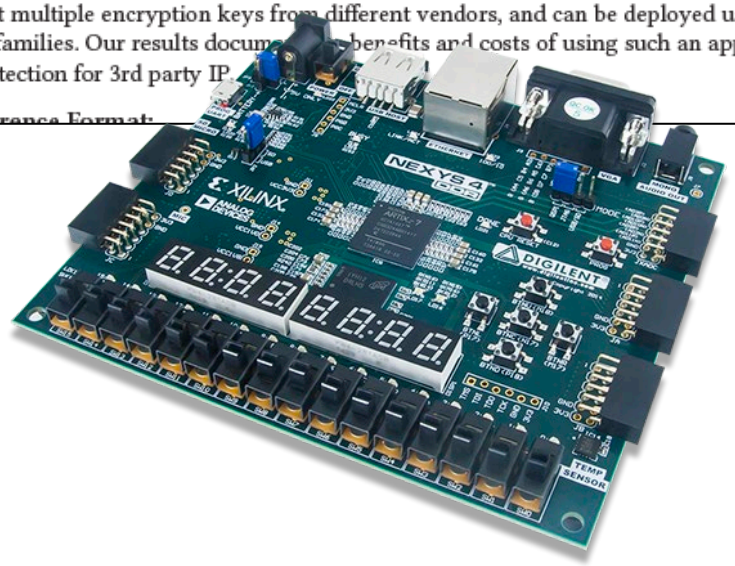
Toward FPGA Intellectual Property (IP) Encryption from Netlist to Bitstream

DANIEL HUTCHINGS, Brigham Young University, USA
ADAM TAYLOR, Brigham Young University, USA
JEFFREY GOEDERS, Brigham Young University, USA

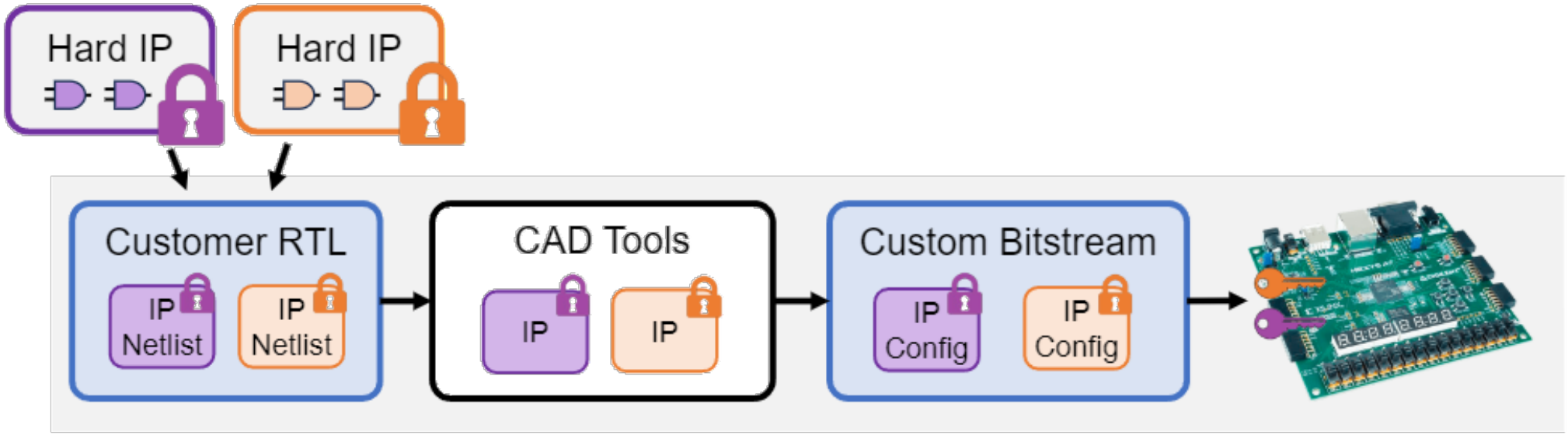
Current IP encryption methods offered by FPGA vendors use an approach where the IP is decrypted during the CAD flow, and remains unencrypted in the bitstream. Given the ease of accessing modern bitstream-to-netlist tools, encrypted IP is vulnerable to inspection and theft from the IP user. While the entire bitstream can be encrypted, this is done by the user, and is not a mechanism to protect confidentiality of 3rd party IP.

In this work we present a design methodology, along with a proof-of-concept tool, that demonstrates how IP can remain partially encrypted through the CAD flow and into the bitstream. We show how this approach can support multiple encryption keys from different vendors, and can be deployed using existing CAD tools and FPGA families. Our results docum[illegible] benefits and costs of using such an approach to provide much greater protection for 3rd party IP.

ACM Reference Format:



IP Encryption Framework



Problems:

1. Requires new CAD tools
   - How can you do CAD on encrypted IP?

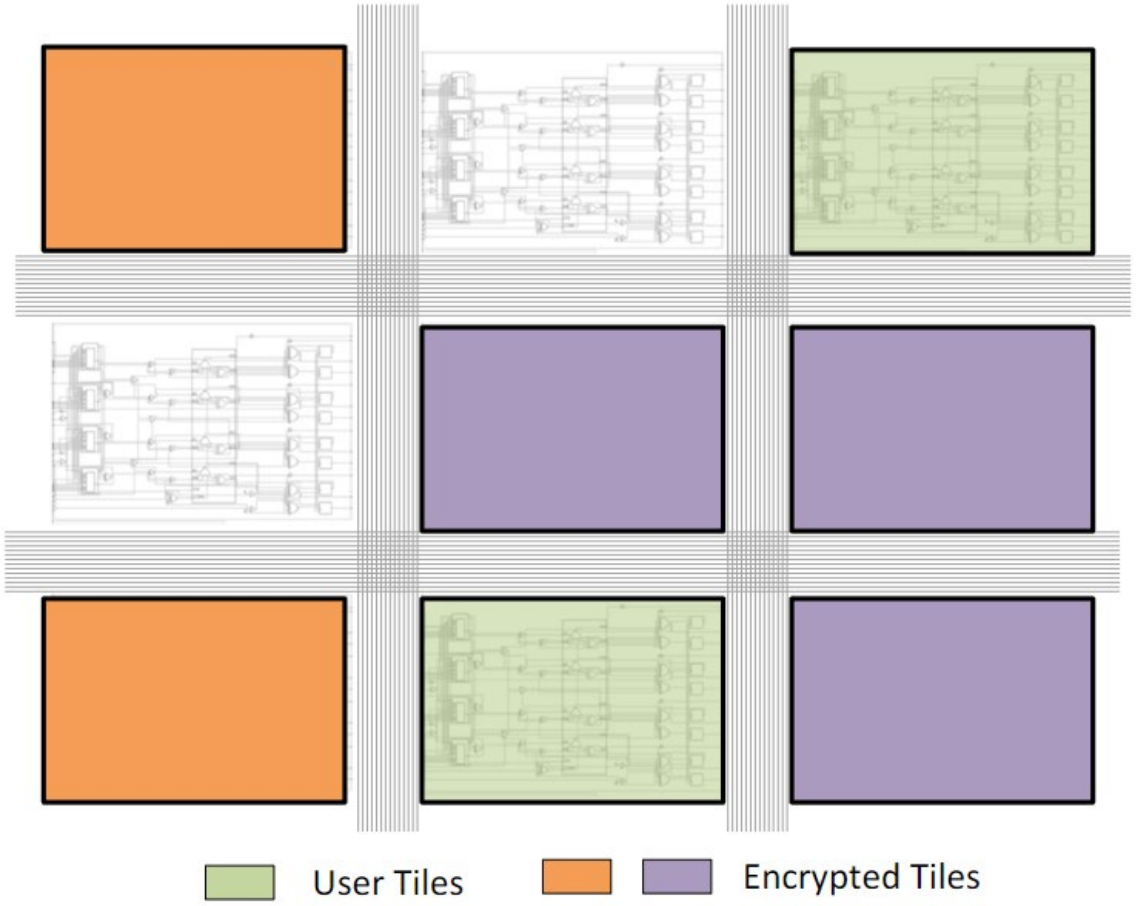
2. Requires new FPGA devices
   - Perform fine-grained decryption during configuration

Encryption Granularity

Not possible to encrypt the entire IP. What can we encrypt and still perform CAD?

Encrypt LUTs

Encrypt Tiles

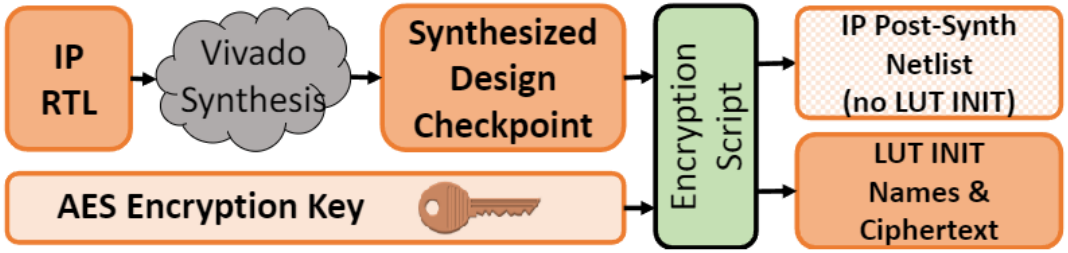


Can we use commercial CAD tools to operate on the (partially) encrypted design?

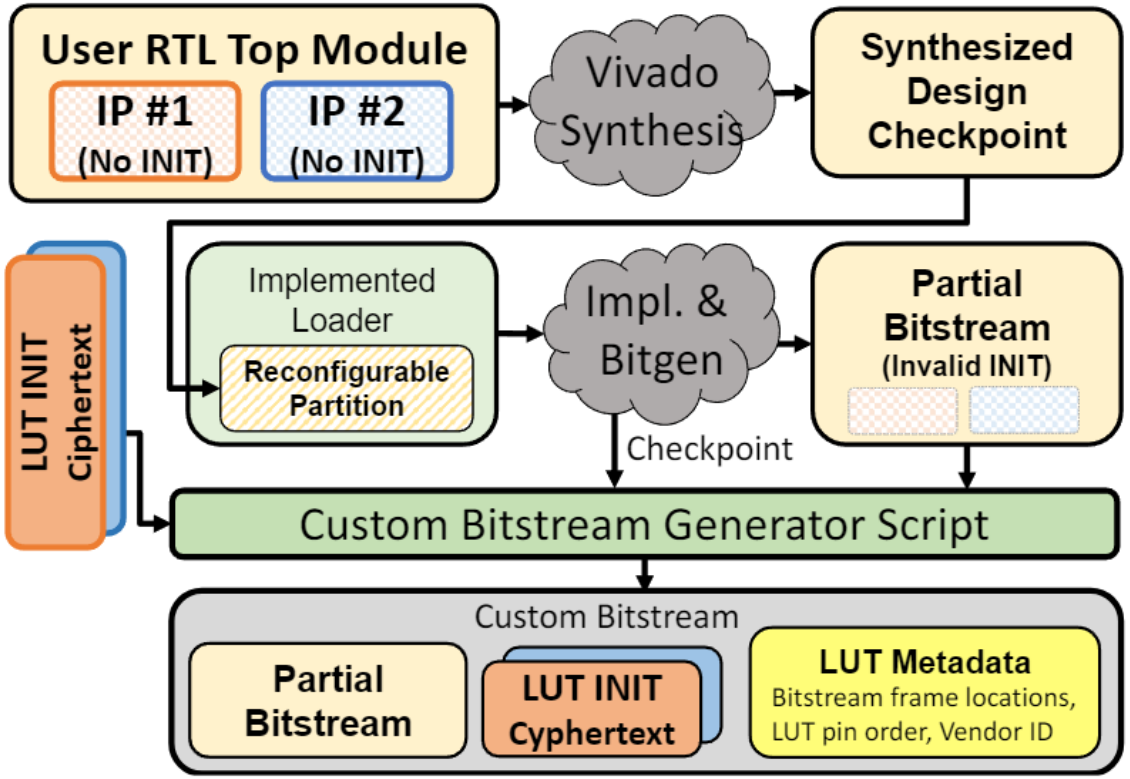
What happens when you lie to your CAD tool about your design?

Demonstrated CAD Flow

IP Vendor Flow:



IP Customer Flow:



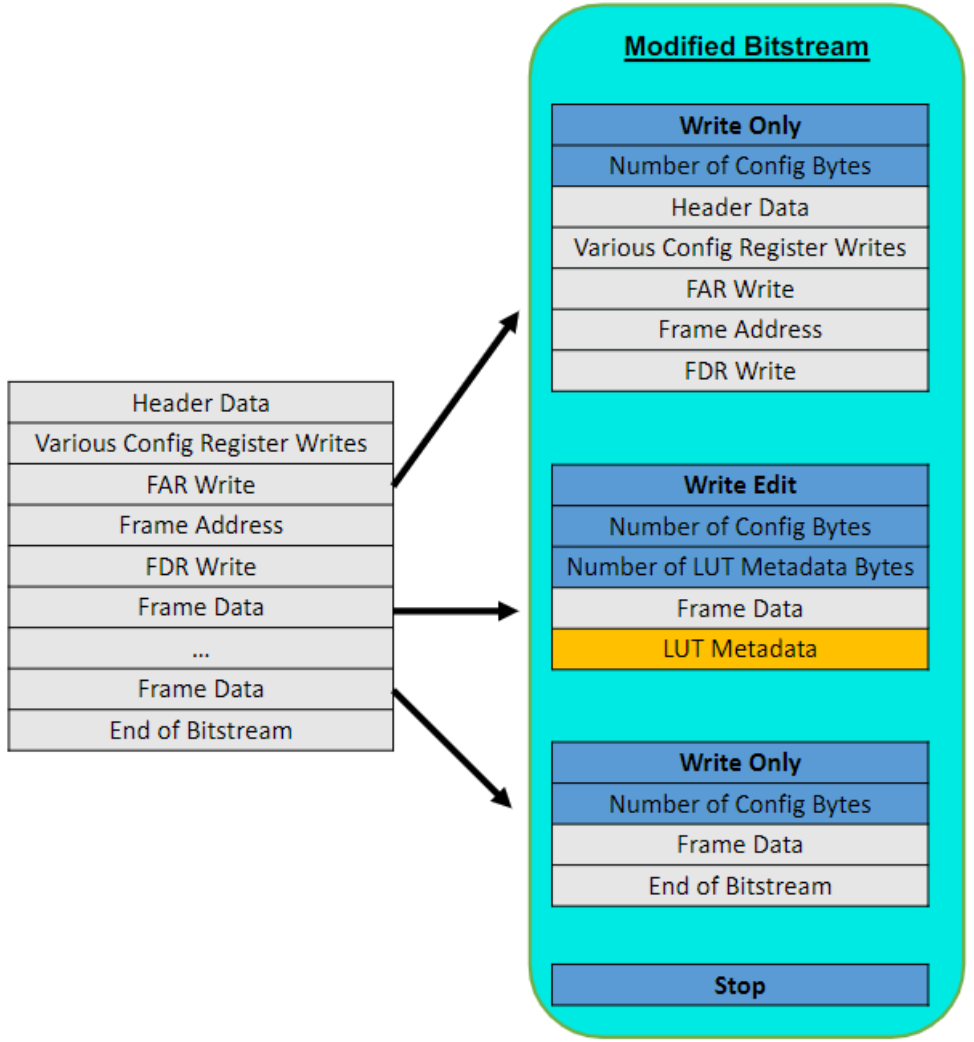
Custom Bitstream: Encrypted Data+

We use a custom bitstream format that contains a mixture of unencrypted User logic and encrypted LUTs.

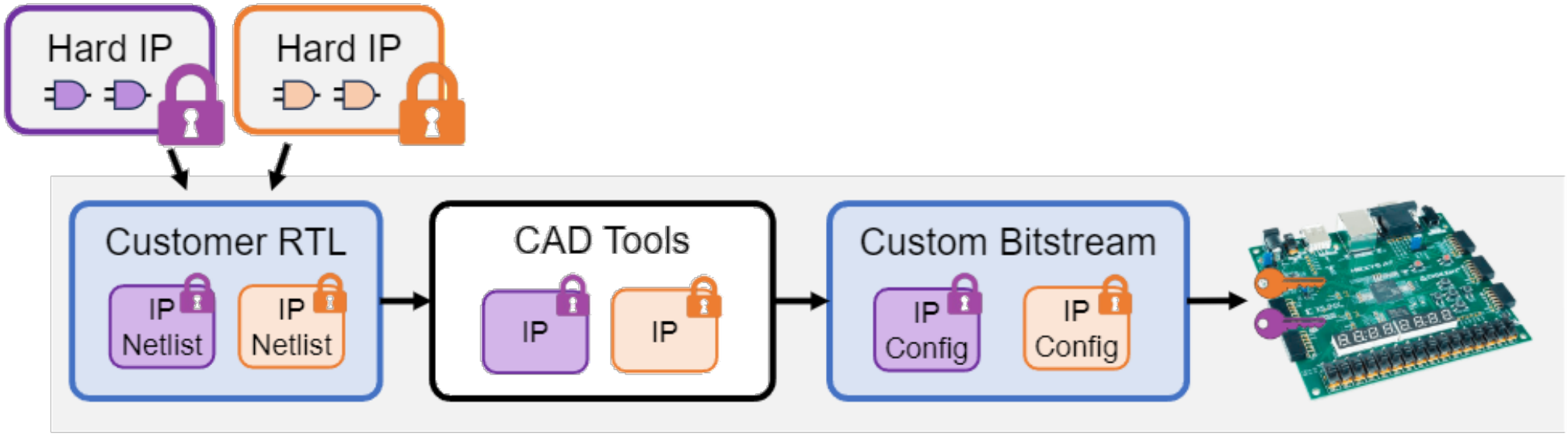
Plus **extra metadata to handle optimizations performed by the CAD tool** on the “fake” (encrypted) logic.

- LUT INIT inflation
- LUT pin reordering
- Fractured LUT combining

These optimizations are tracked, so they can be **“redone” during reconfiguration, post-decryption**.



IP Encryption Framework



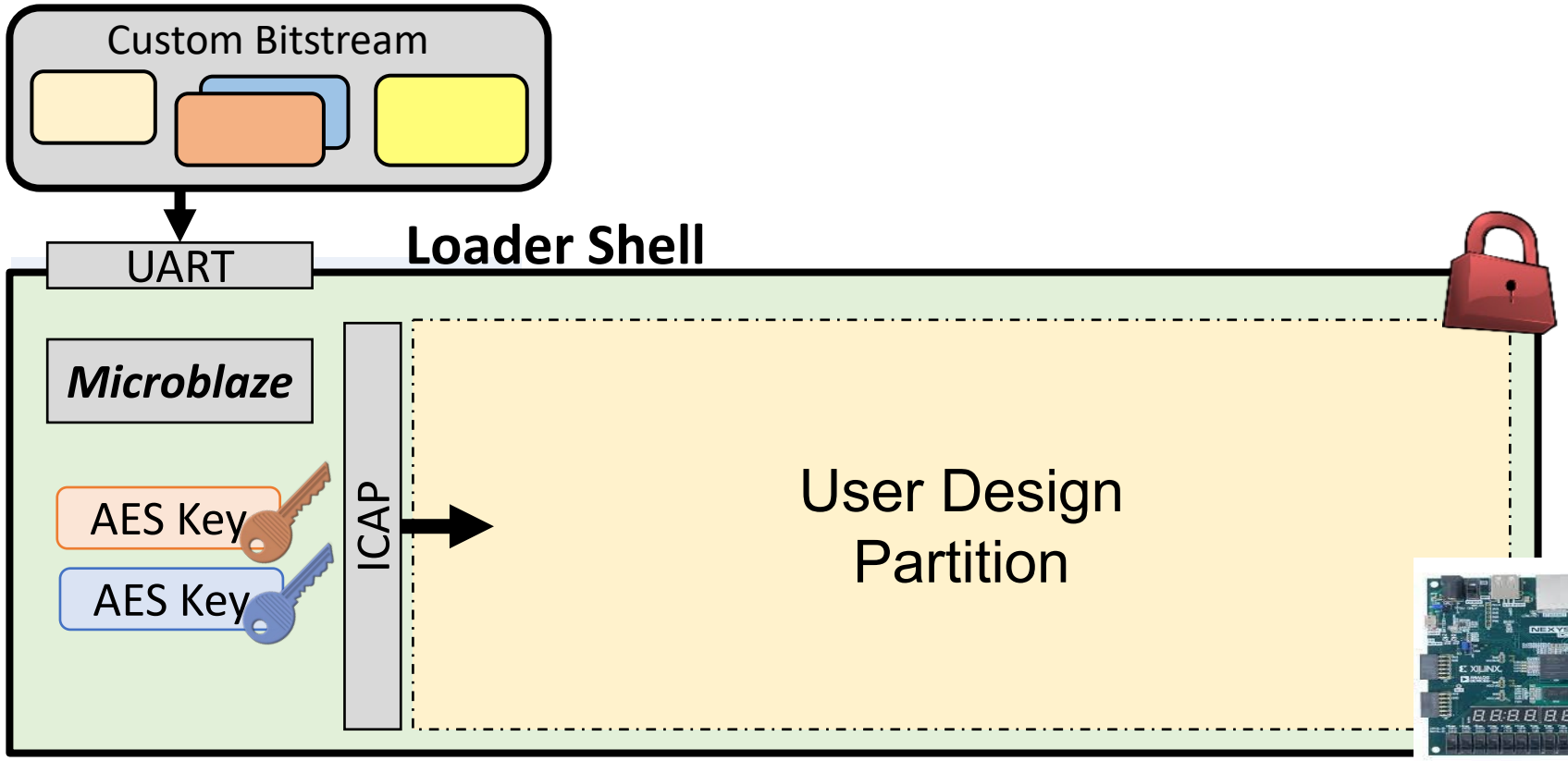
Problems:

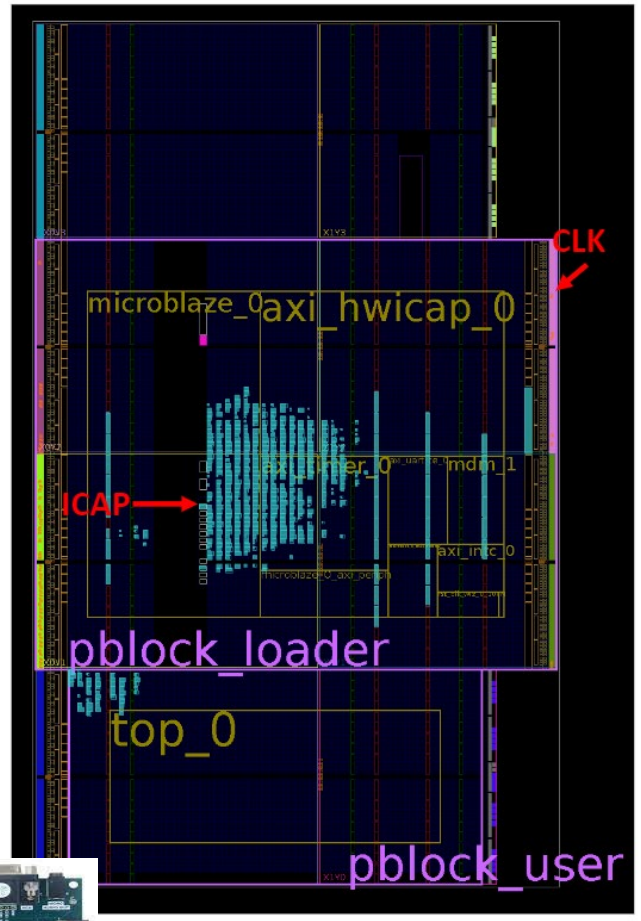
1. Requires new CAD tools
   - How can you do CAD on encrypted IP?

2. Requires new FPGA devices
   - Perform fine-grained decryption during configuration

Enhanced Configuration Circuitry

- New configuration circuitry is required to hold per-vendor decryption keys, and perform fine-grained decryption
- We implement this as a “Static Shell”, with all user logic constrained to a reconfigurable partition (e.g. similar to cloud FPGA usage)





Key Management

User must be prevented from looking inside the FPGA loader shell
(which contains the IP encryption keys)

FPGA device must come from a trusted key holder with pre-set eFUSE values to decrypt Loader Shell, prevent readback, etc.

CFG_AES_Only: Forces the use of AES key stored in eFUSE and disables device readback. This bit must be set as the FPGA must only accept the Loader bitstream, and not another bitstream created by the user to access the IP decryption keys.

AES_Exclusive: Disables partial reconfiguration from external configuration interfaces but still allows partial reconfiguration via the ICAP. This bit must also be set for the same safety concern as the previous bit.

W_EN_B_Key_User: Disables programming of AES key. This bit must also be set to prevent the IP user from overwriting the AES key.

R_EN_B_Key, R_EN_B_User: Disables reading and reprogramming of AES key.

W_EN_B_Cntl: Disables any further changes to the eFUSE registers.

This exposes new attack vectors (e.g. side channel attacks, starbleed-like attacks, etc.)

Overall, we have made obtaining encrypted IP netlists much more challenging than with existing tools.

Project 1: Verifying bitstream-to-netlist equivalence

Reilly McKendrick, Keenan Faulkner, and Jeffrey Goeders, “Assuring Netlist-to-Bitstream Equivalence using Physical Netlist Generation and Structural Comparison,” International Conference on Field-Programmable Technology, Dec 2023.

Project 2: Protecting Encrypted IP

Daniel Hutchings, Adam Taylor, Jeffrey Goeders, “Toward Intellectual Property (IP) Encryption from Netlist to Bitstream”, ACM Transactions on Reconfigurable Technology and Systems, *to be published*, 2024.

Questions?

Contact Me: Jeff Goeders, jgoeders@byu.edu

Trust

What trusted tools do we require?

1. A bitstream-to-netlist tool.
   - We use an open-source tool (fasm2bels), but a capable organization (e.g. state actor) may elect to make their own.
2. A netlist manipulation tool.

Don’t we trust the commercial CAD tool when asking for the set of implementation transformations? What if it maliciously lied?

No! An incorrect set of transformations will simply cause our tool to fail to verify equivalence.

Our netlist transformations don’t change the design functionality, so it would be impossible to hide malicious design changes through a false set of transformations.

A malicious list of transformations may cause a false negative comparison, but it’s not possible to induce a false positive comparison.

Limitations

Can we support any design? Why not try out very large designs?

- The 3rd party tool for bitstream to netlist conversion does not support all device features.
- Larger designs are more likely to implement these unsupported features.
- Our work is a proof-of-concept. A production tool would need to have a complete bitstream-to-netlist tool.

Can this work for other FPGA vendors?

- Using other vendors requires having similar documentation of the bitstream in order to produce the reversed netlist.
- CAD tool must expose set of design transformations made during implementation.