Jebait !

HLock: Locking IPs at the High-Level Language

Rafid M., Roshanak M., Mark T. and Farimah F

Design Automation Conference(DAC) 2021

March 18, 2022

Presented by
Akshay Gopalakrishnan

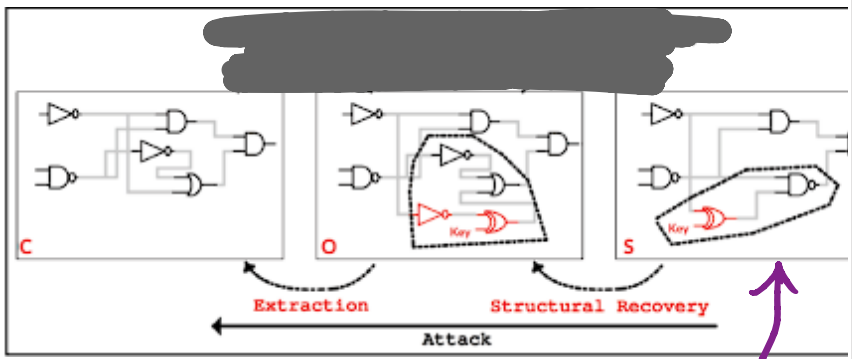
Authors?

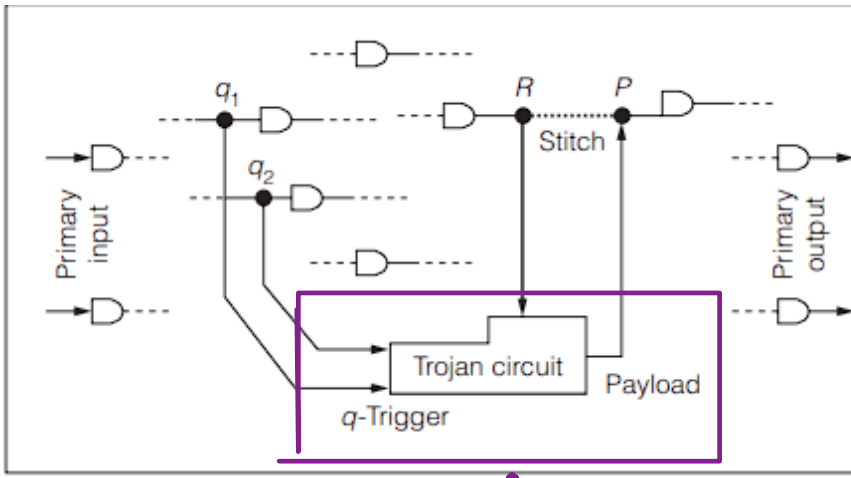
- Cybersecurity research group at University Of Florida
- Farimah and Mark (professors.)
- Rafid and Roshanak (PhD students.)

Outline

- Security ! ⇝ of H/W Synthesis !
- Security from what ?
- Remedy ? "Lock" parts of the code.
- Lock at High Level description to avoid attackers from succeeding (resiliency).
- Results

Security Need

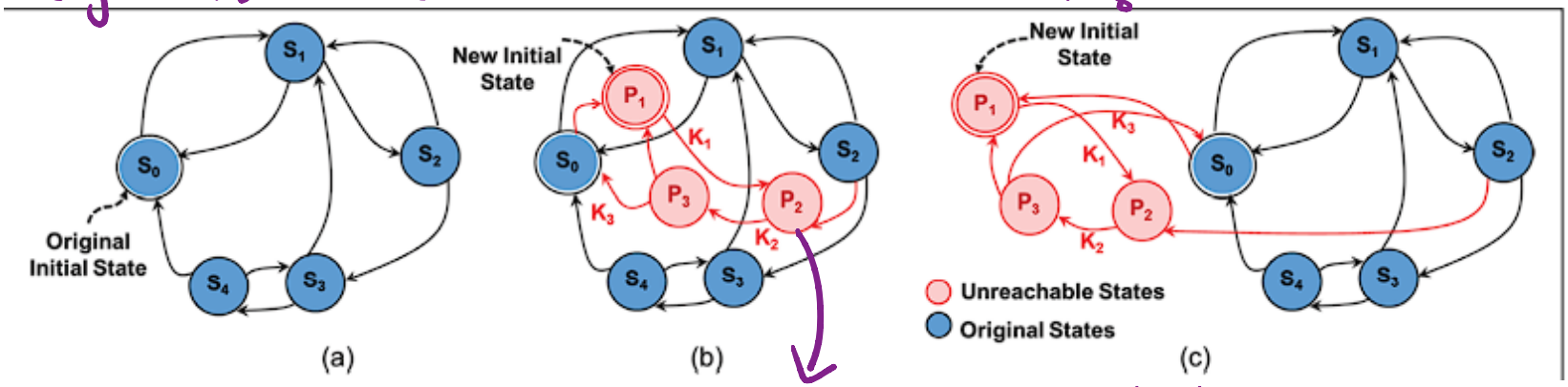




- Intellectual Property (IP) blocks of code.
- IP blocks used for Hardware synthesis.
- Attacks - eg: Hardware Trojans, Reverse Engineering, etc.

Security Measures: Locking/Obfuscation

eg: At State Machine level !



Add redundant states.

- Modify parts of the hardware specification at the RTL/netlist layer.
- The parts work correctly only with another extra input being correct.
- This way, "locking" of IP blocks can be achieved.

Problem?

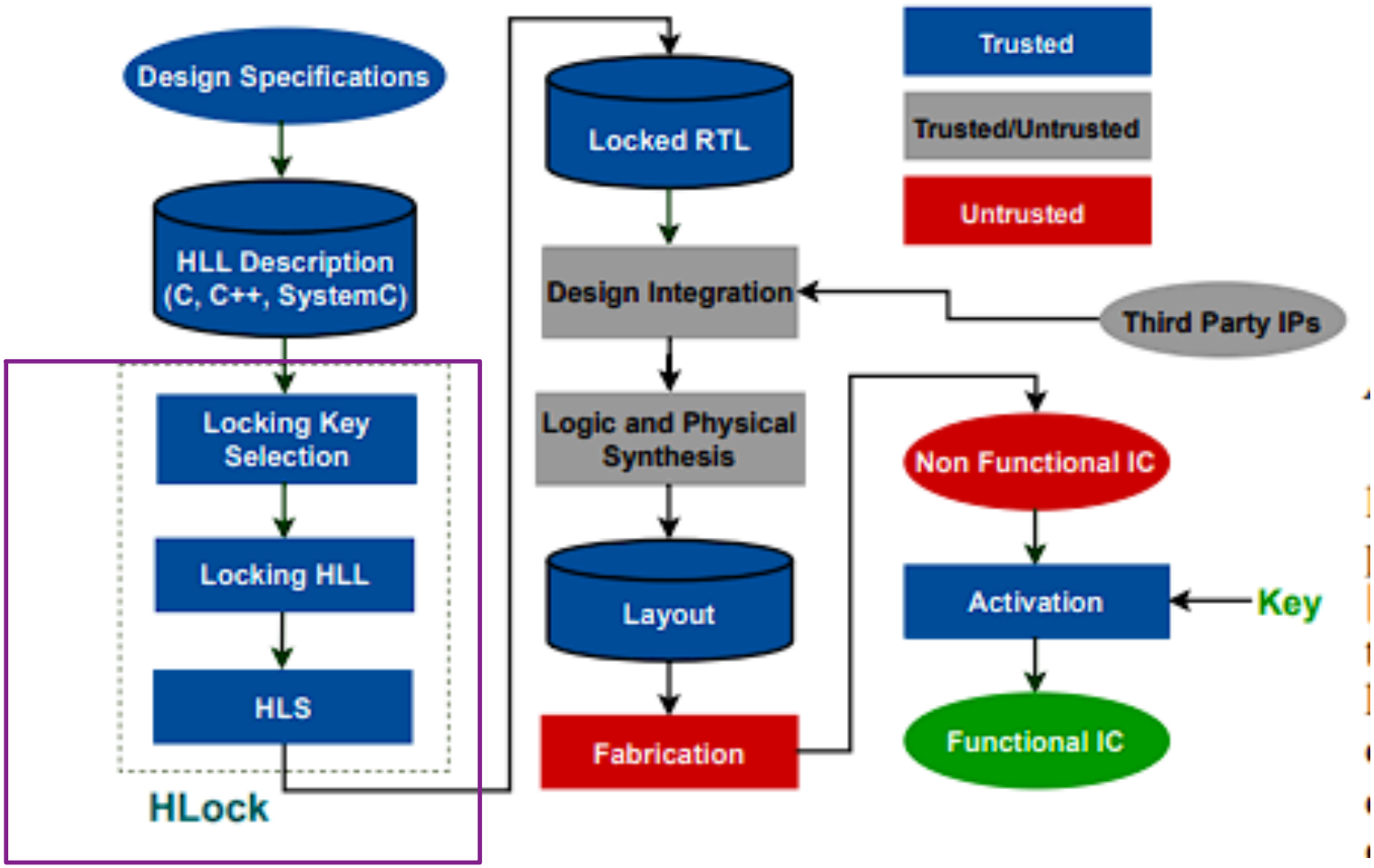
- RTL/netlist layer security not resilient enough.
- Obfuscating constant values and branches of RTL are hard to do. → Easy to detect.
- SAT based/ Machine learning based attacks can easily extract the original design.

Not resilient enough!!

Proposed Solution

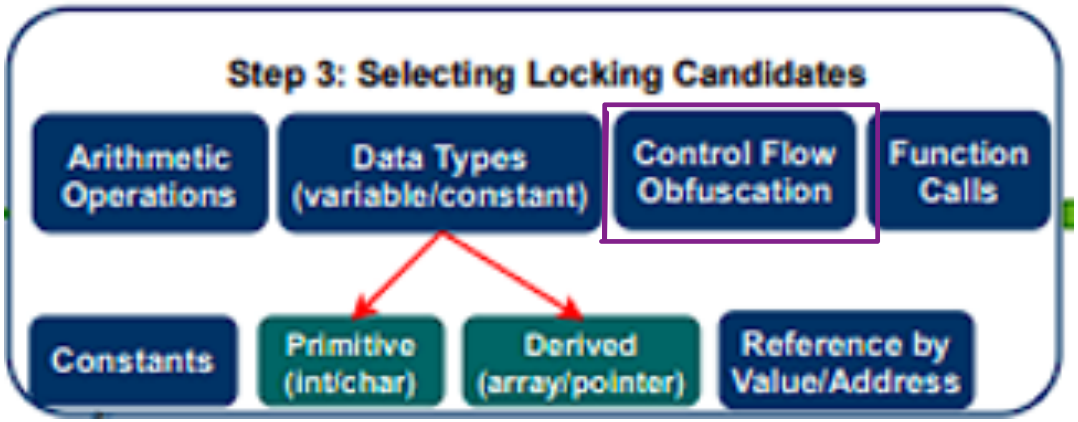
- Perform locking/obfuscation at HLS level (C/C++ like) design.
- Previous approach exists in these lines, but do not measure resilience to attack and has more overhead.

Outline

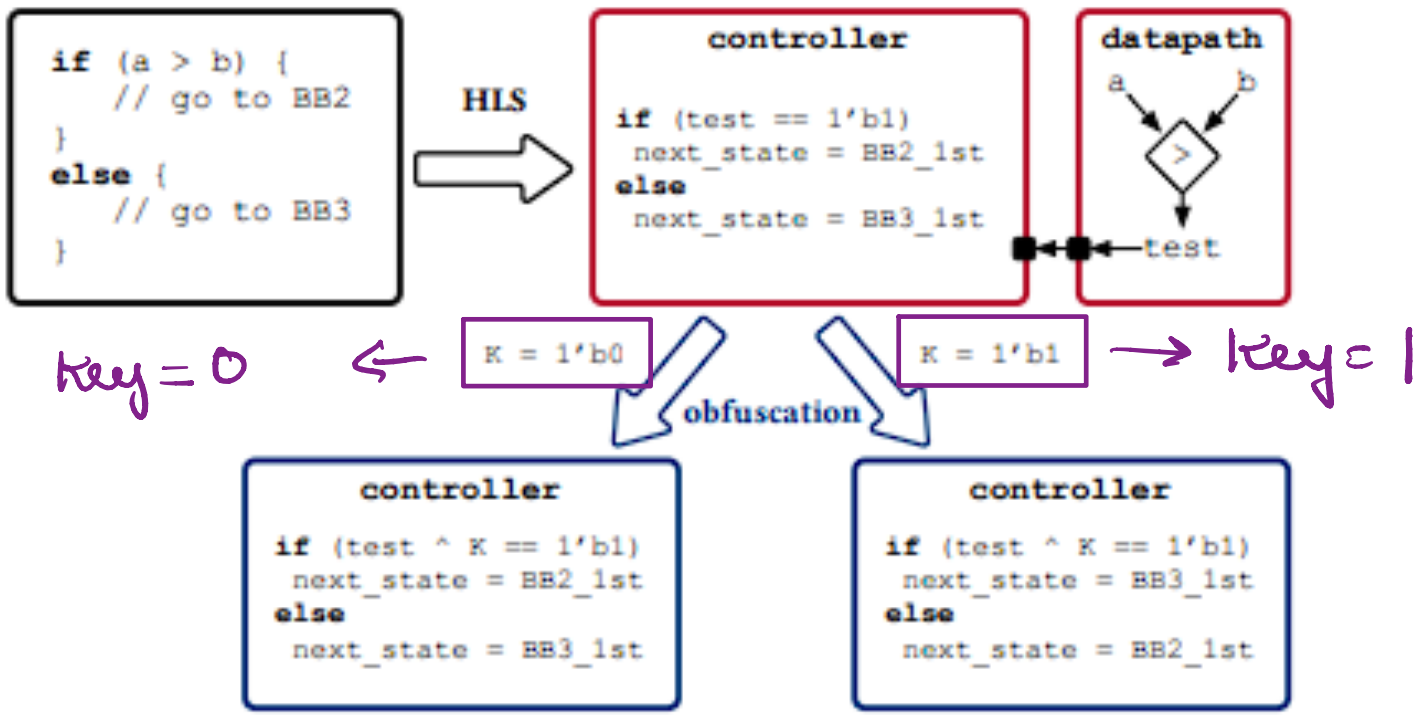


↳ Proposed Solution

Locking Different Candidates



Branch Obfuscation

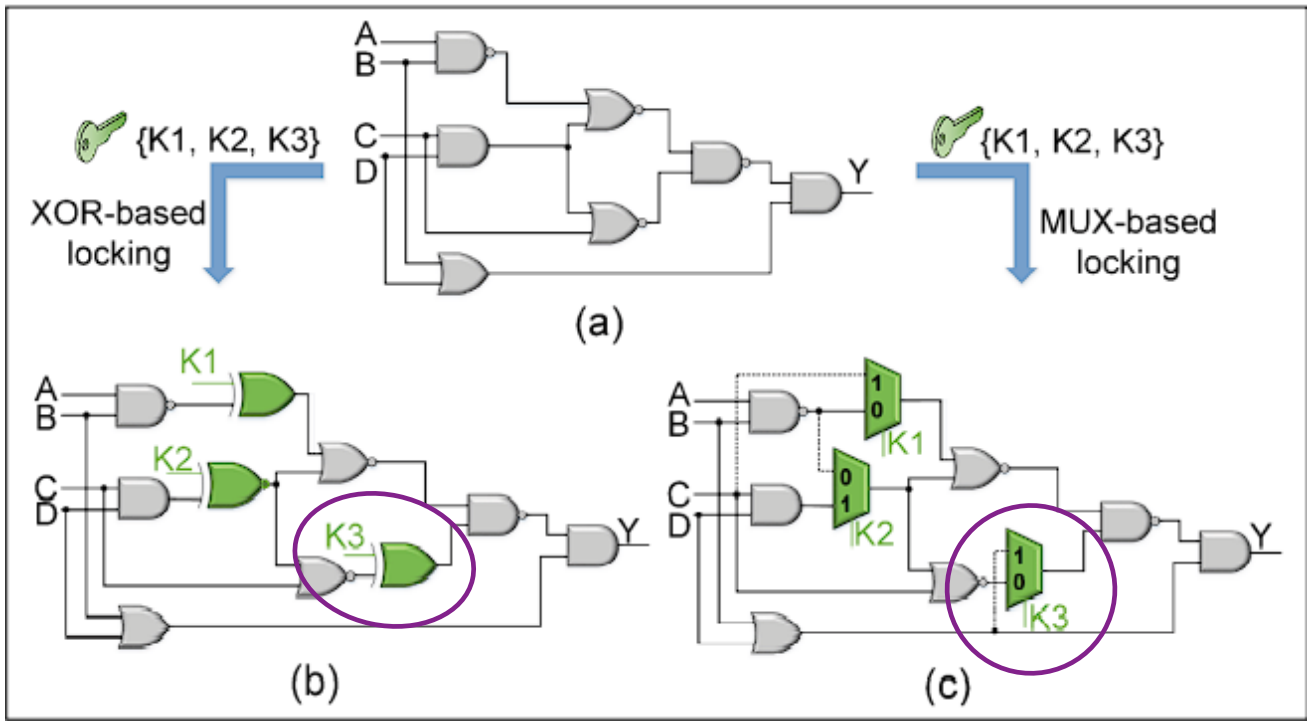


Function Obfuscation

Own code sample here.

```
int add (int a, int b)
        { hock/Obfuscate
int add (int a, int b, int Key )
```

Constant Obfuscation



Identifying Optimal Lock Key Size

Model as ILP problem.

Key size for candidate 2c

$$\gamma_{1c} \times L_{1c} + \gamma_{2c} \times L_{2c} + \ldots + \gamma_{mc} \times L_{mc} \geq Res_{spec} \quad (1)$$

Resiliency Value

Max Resiliency

$$\alpha_{1c} \times L_{1c} + \alpha_{2c} \times L_{2c} + \ldots + \alpha_{mc} \times L_{mc} \leq Ov_{spec} \quad (2)$$

Area value

Max Area

Whole setup

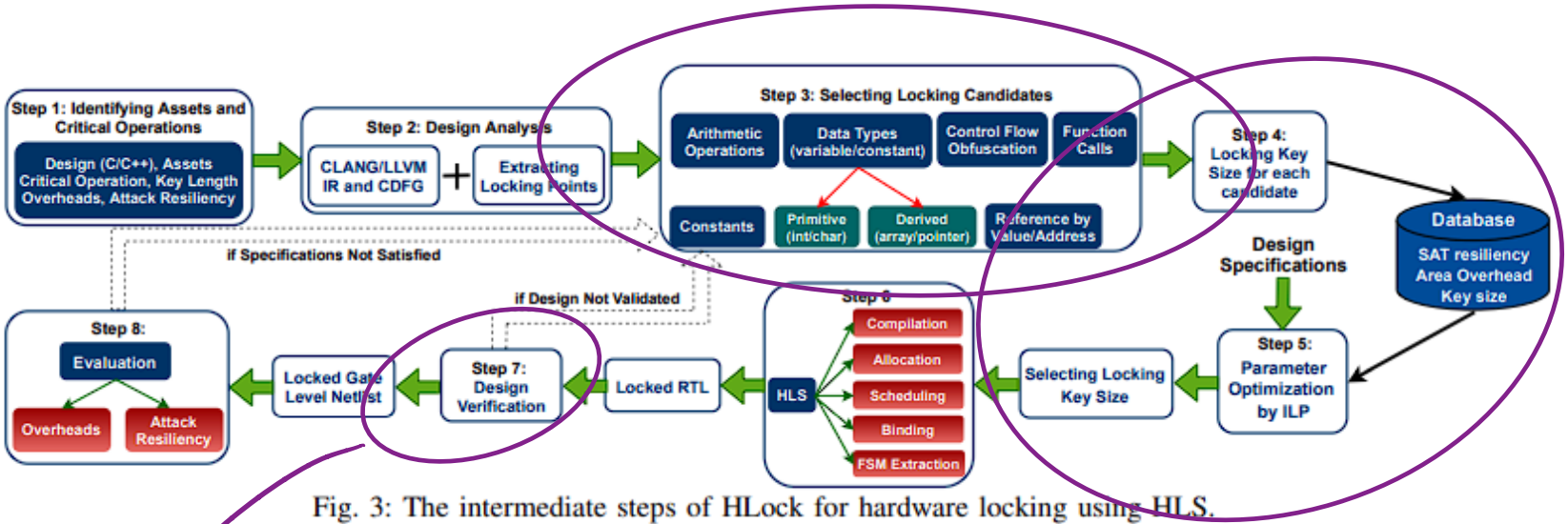
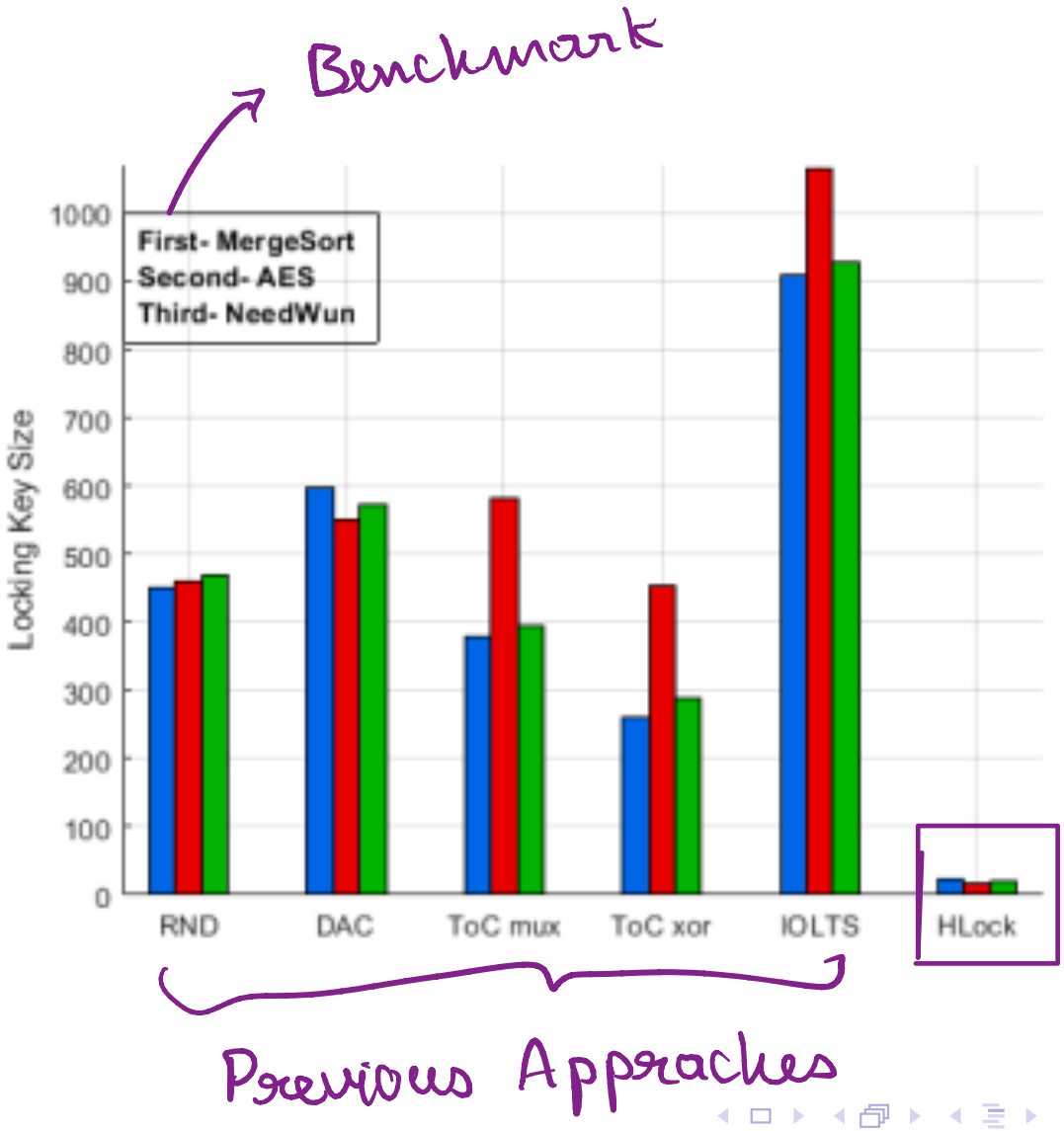


Fig. 3: The intermediate steps of HLock for hardware locking using HLS.

Design should give correct output only for correct keys!

Lock Key Size compared to Previous Approaches



Power consumption and SAT Resiliency

Locking Type	Mergesort	
	Power Overhead	SAT Resiliency
inserts XOR and XNOR gates at randomly chosen locations (RND) [20]	69.09%	10.75s
inserts XOR/XNOR gates carefully to avoid fault-analysis attack (DAC) [19]	103.21%	190.20s
Maximizes HD between correct and incorrect outputs by MUX (ToC mux) [21]	42.10%	1.34s
Maximizes HD between correct and incorrect outputs by XOR (ToC xor) [21]	82.30%	19.34s
Minimizes low controllability locations by inserting AND, OR (IOLTS) [29]	14.67%	2.90s
HLock (Proposed Framework)	**7.84%**	**1915s**

ML Resiliency

Benchmark Designs	Accuracy (%) for Locking Types				
	TOCm'13 [21]	IOLTS'14 [29]	SARLock [22]	Mux2 [30]	HLock
MergeSort	96.66	100	100	92.27	68.18
AES	97.22	100	100	93.82	62.50
NeedWun	98.86	99.32	100	92.74	65.87
Avg.	*97.58*	*99.77*	*100*	92.95	***65.51***

A few drawbacks

- Resiliency is highly reliant on optimizations done by HLS tools to locked design.
- Comparison of results are with previous RTL/Netlist layer locking (not the previous work on HLL layer).
- Lack statistics about time taken to lock the design (potentially much slower than previous approaches).

Thank you

Questions?