ACM/IEEE CODES + ISSS 2017, Seoul, South Korea

# Efficient Control-Flow Subgraph Matching for Detecting Hardware Trojans in RTL Models

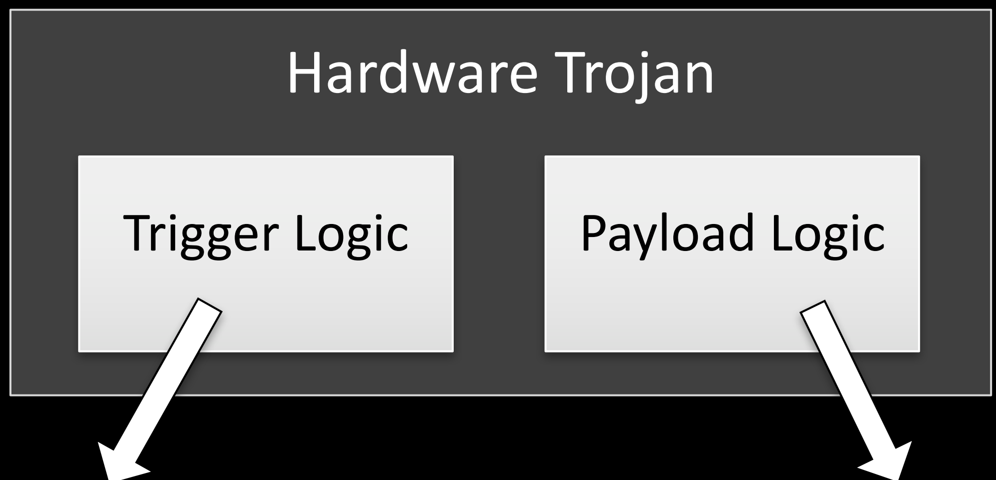
**L. Piccolboni**[1,2], A. Menon[2], and G. Pravadelli[2]

[1] Columbia University, New York, NY, USA

[2] University of Verona, Verona, Italy

# Hardware Trojans

- A Hardware Trojan is defined as a malicious and intentional alteration of an integrated circuit that results in undesired behaviors




activates the malicious behavior under specific conditions

implements the actual malicious behavior

# Hardware Trojans

## Limitations in Current Methodologies

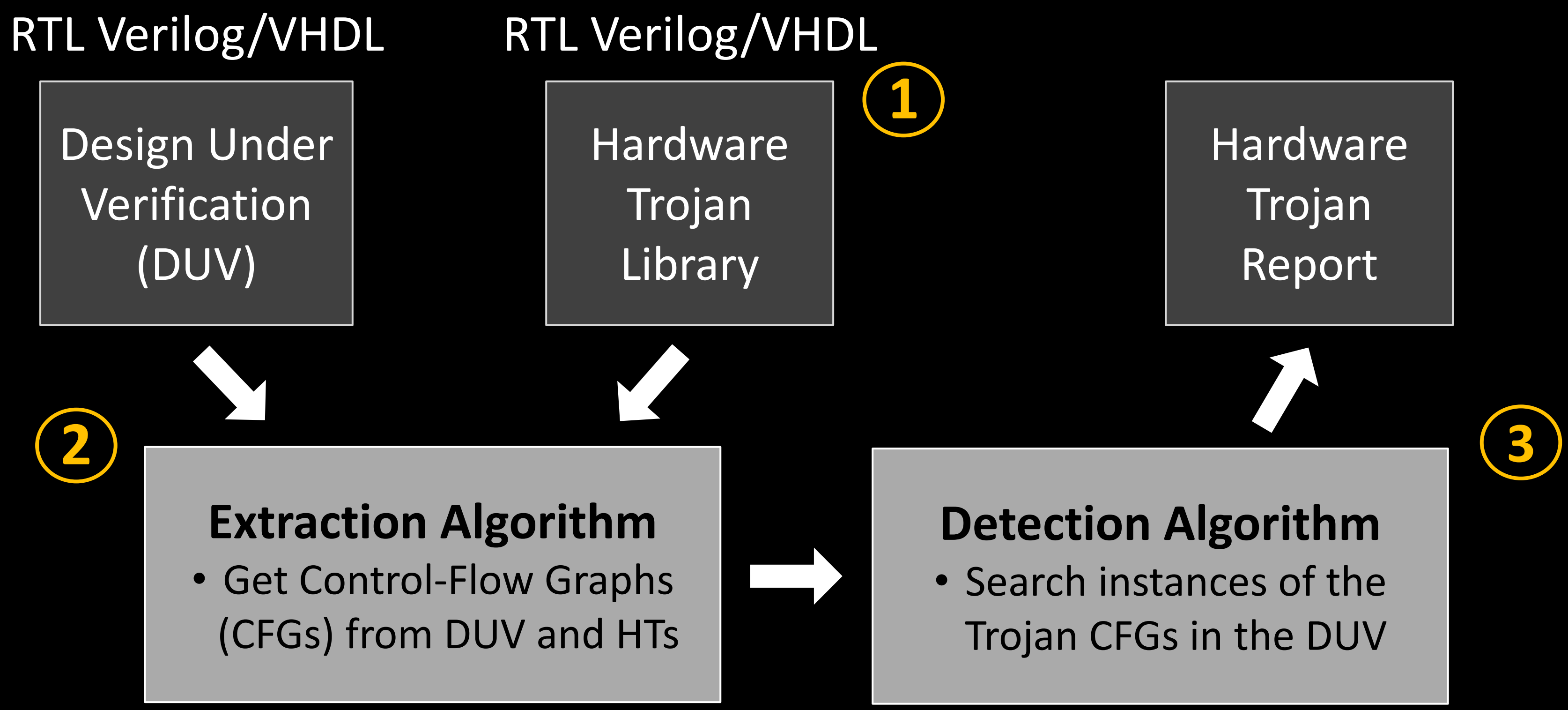
- Several methodologies have been proposed to detect Trojans at Register-Transfer Level (RTL)

- Nevertheless, there are still some limitations:

  1. Manual effort from designers is required
  2. They focus on a specific type of threat, e.g., a particular payload or a trigger

# Contributions

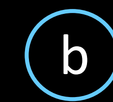
- We propose a verification approach based on a Control-Flow Subgraph Matching Algorithm



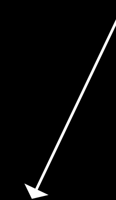
# Background

## Control-Flow Graphs (CFGs)

- We build a CFG for each process of the DUV/HT
  - basic block (node) = it is a sequence of instructions without any branch



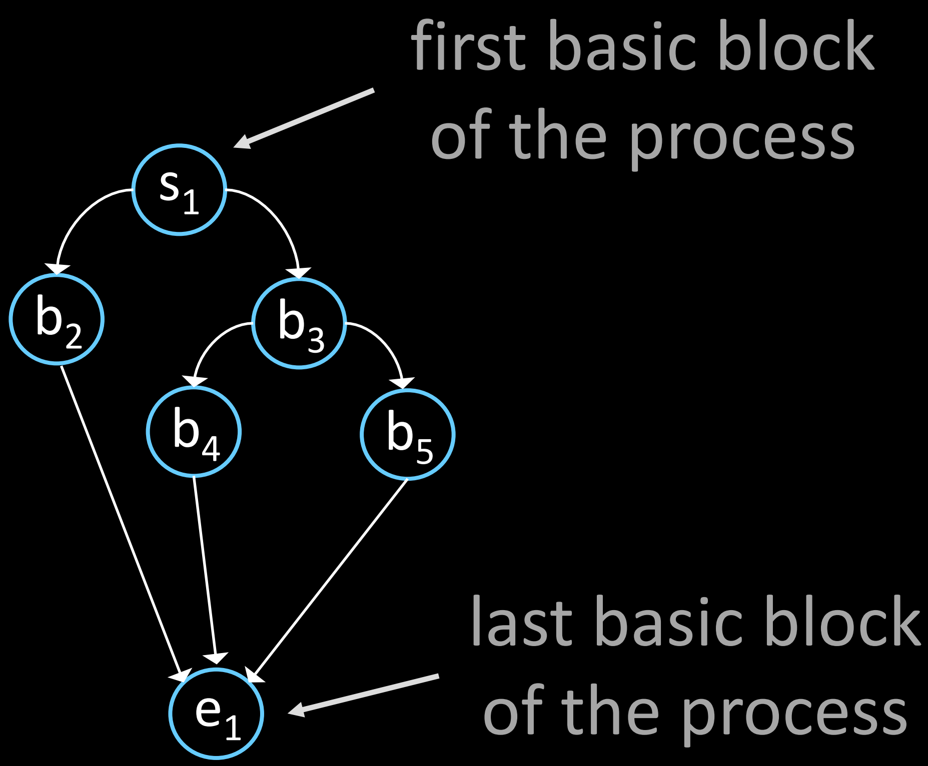

  - edge = connects the block $b_1$ with $b_2$ if the block $b_1$ can be executed after $b_2$ in at least one DUV/HT executions



# Background

## Control-Flow Graphs (CFGs)

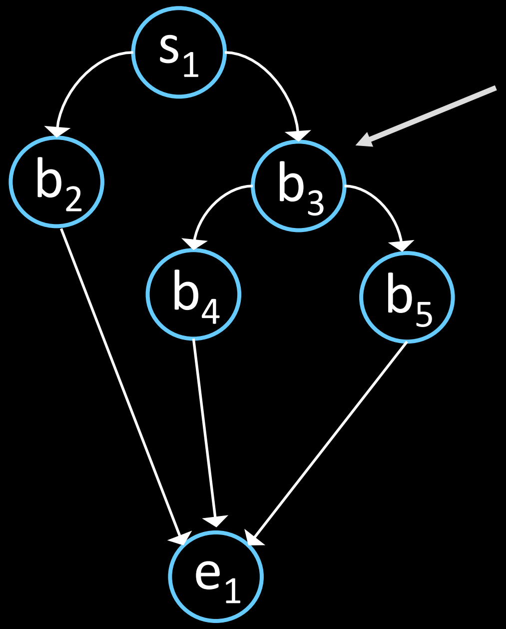
- We build a CFG for each process of the DUV/HT



# Background

## Control-Flow Graphs (CFGs)

- We build a CFG for each process of the DUV/HT



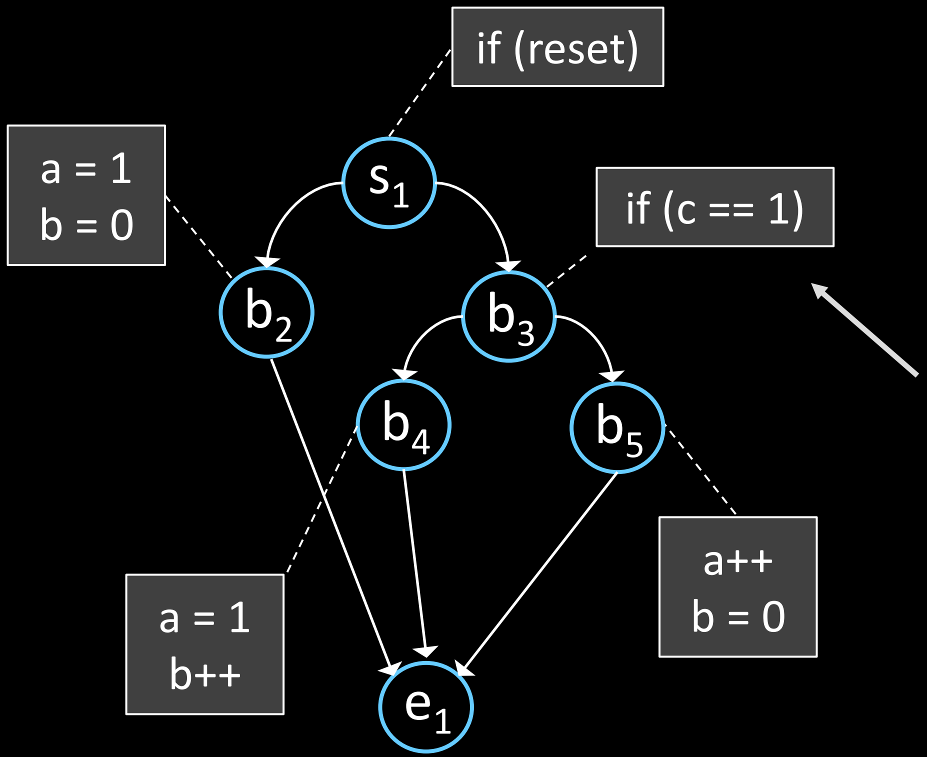

Branch rule:

- left if true
- right if false

# Background

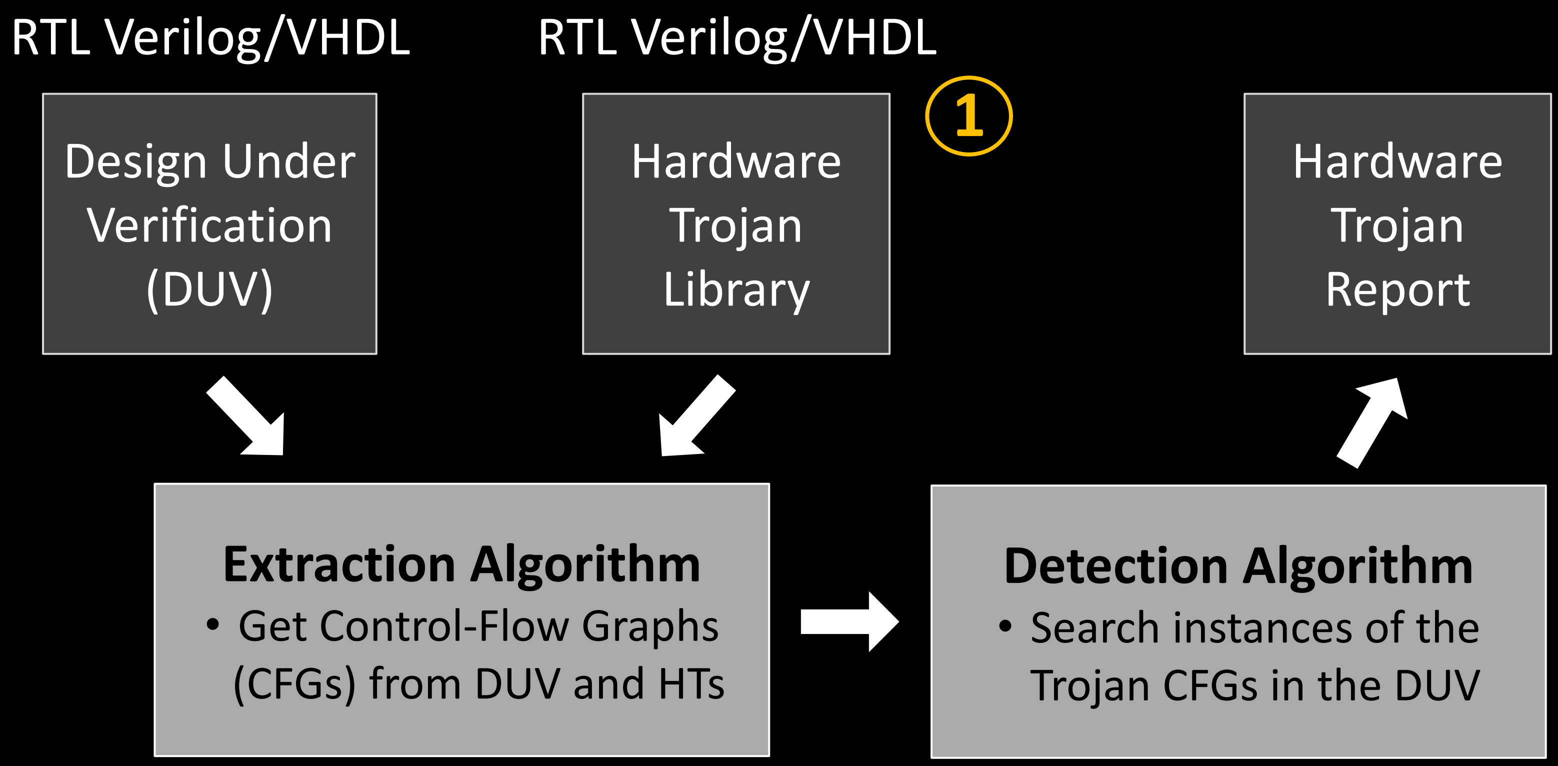
## Control-Flow Graphs (CFGs)

- We build a CFG for each process of the DUV/HT




code associated with the basic blocks

# Hardware Trojan Library



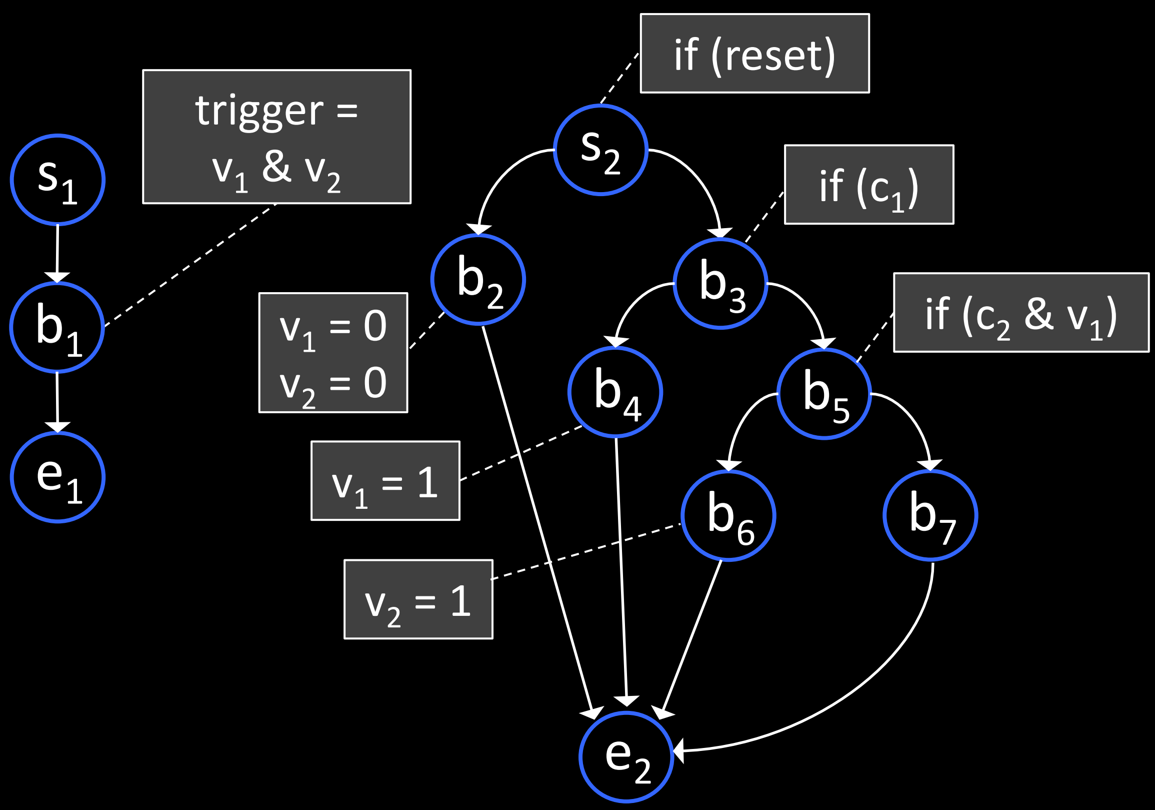
# Hardware Trojan Library

- We defined a **Hardware Trojan (HT) Library** that includes the RTL implementations of known HT triggers and their camouflaged variants

# Hardware Trojan Library

## Trigger #1: Cheat Codes

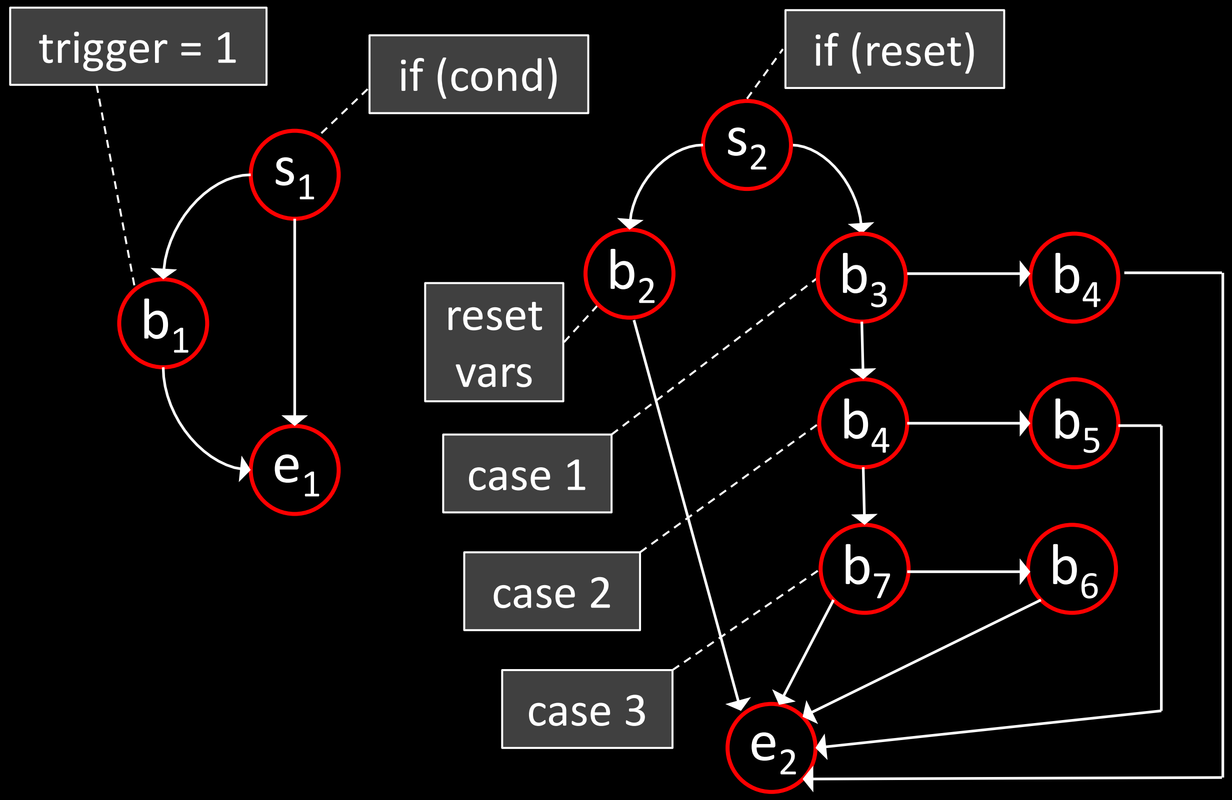
- A cheat code is a value (or sequence of values) that triggers the payload when observed in a register



# Hardware Trojan Library

## Trigger #2: Dead Machines

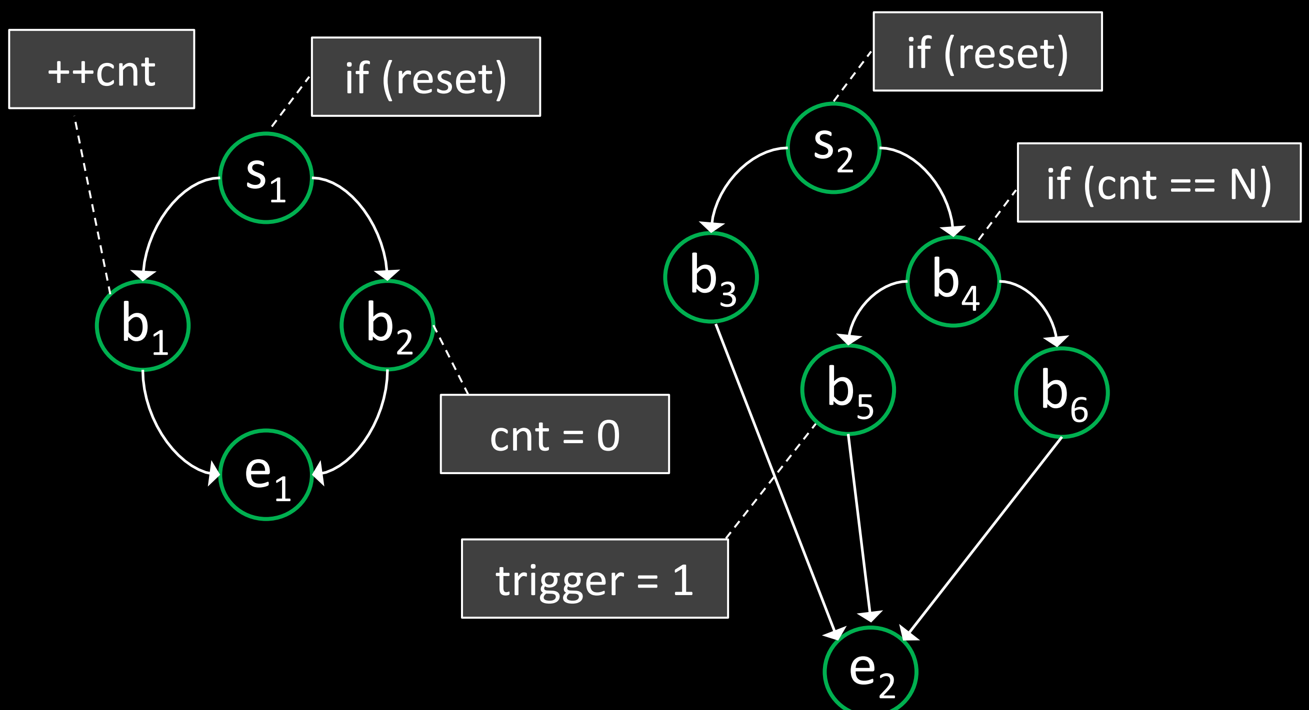
- A dead machine code triggers the payload when specific state-based conditions are satisfied



# Hardware Trojan Library

## Trigger #3: Ticking Timebombs

- A ticking timebomb triggers the payload when a certain number of clock cycles has been passed



# Hardware Trojan Library

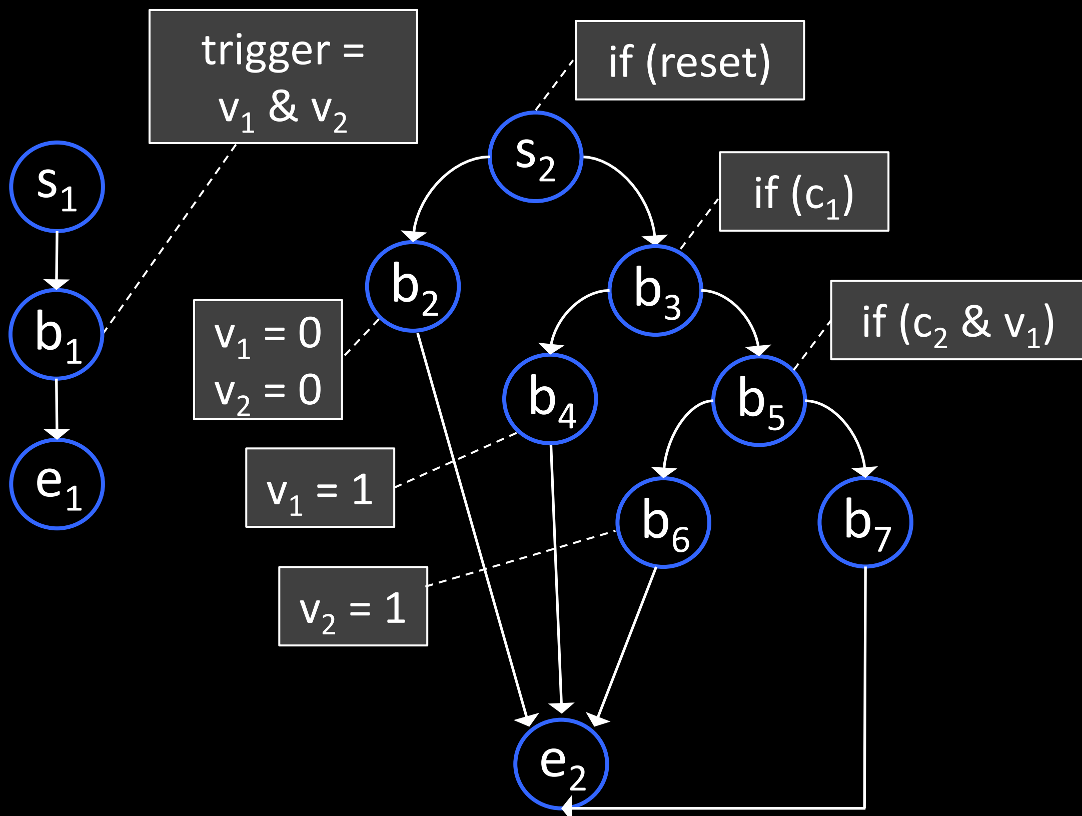
## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



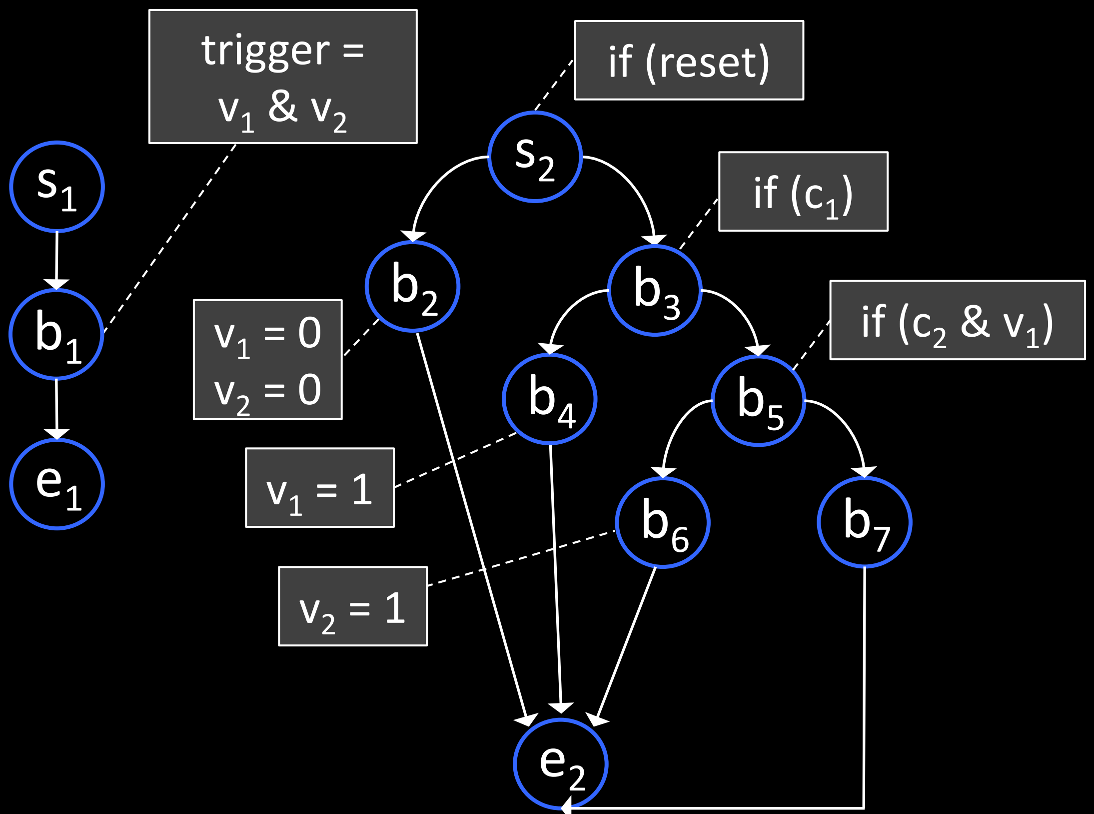

Extension directives:

1. parametrizable 1

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



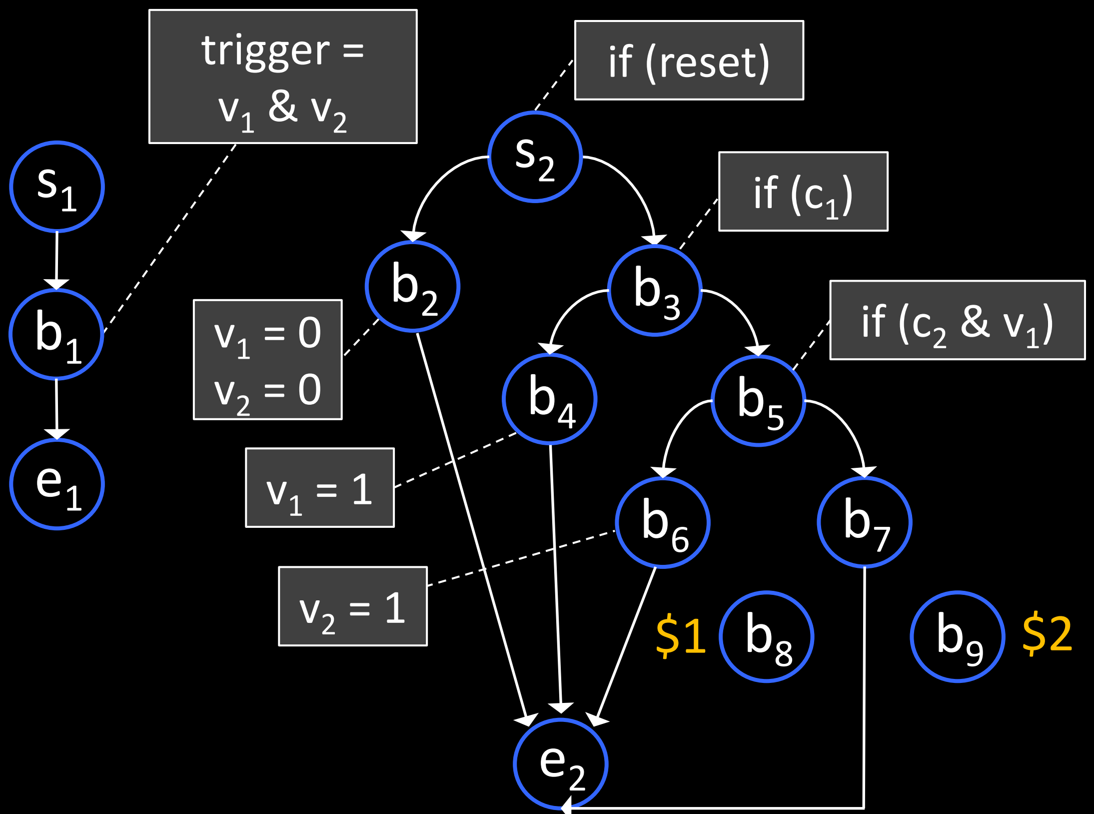

Extension directives:

1. parametrizable 1
2. bound-number 10

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



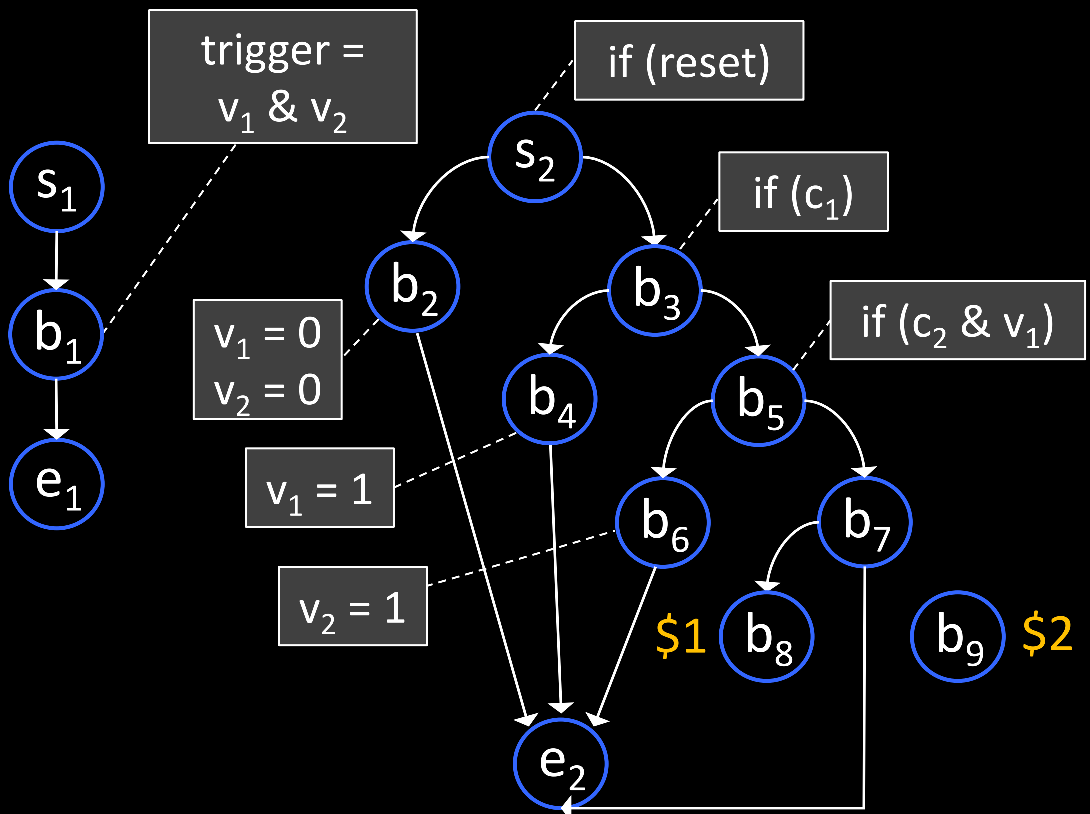

Extension directives:

1. parametrizable 1
2. bound-number 10
3. add-basic-blocks 2

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



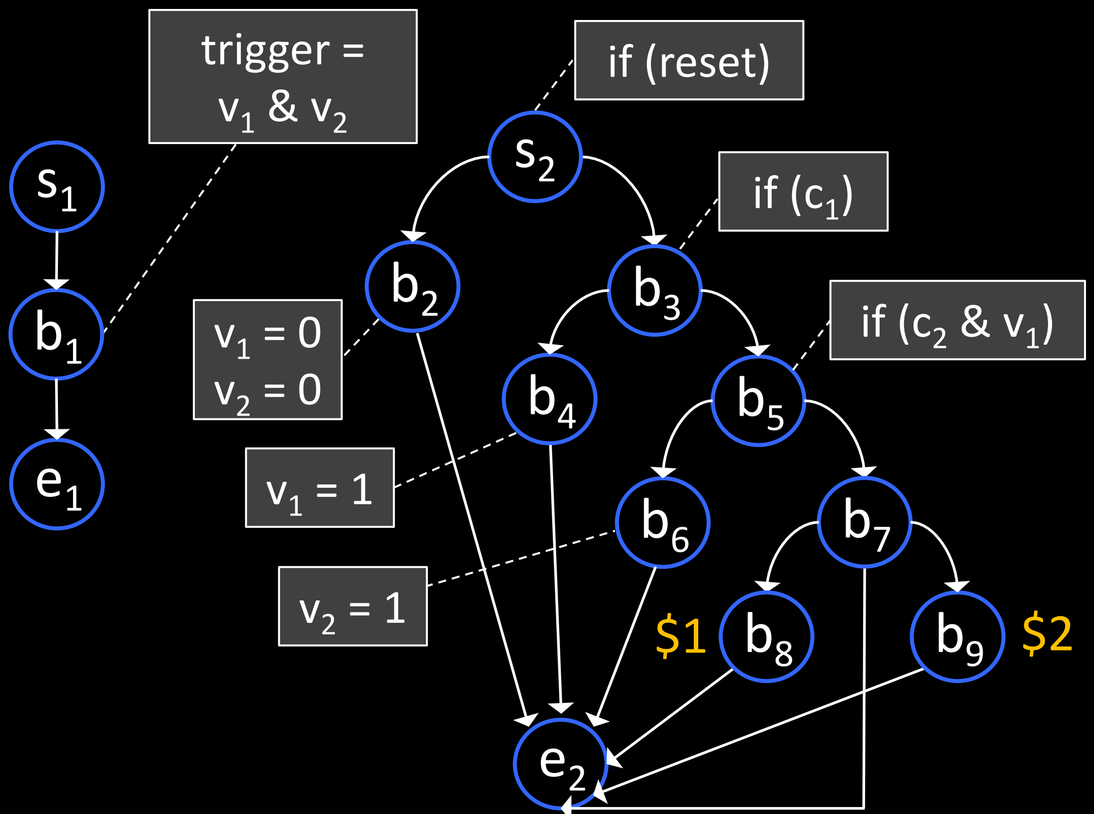

Extension directives:

1. parametrizable 1
2. bound-number 10
3. add-basic-blocks 2
4. add-edge ($b_7$, $1)

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



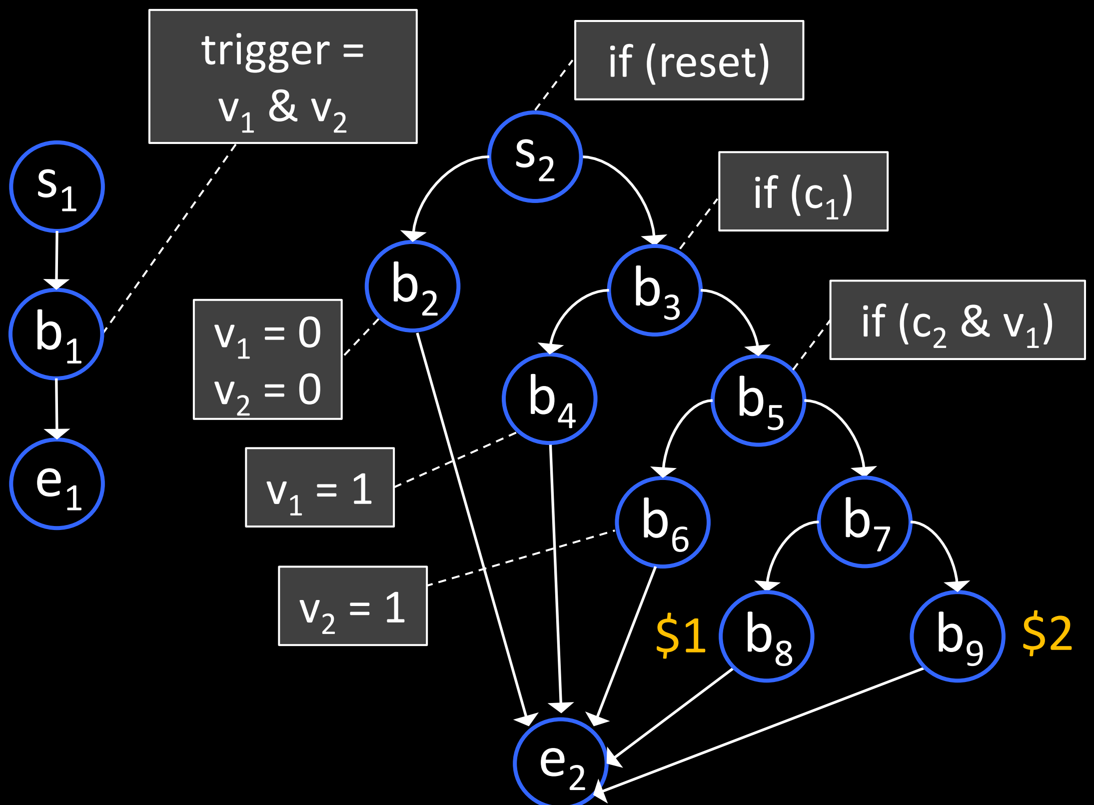

Extension directives:

1. parametrizable 1
2. bound-number 10
3. add-basic-blocks 2
4. add-edge ($b_7$, \$1)
5. add-edge ($b_7$, \$2)
6. add-edge (\$1, $e_2$)
7. add-edge (\$2, $e_2$)

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



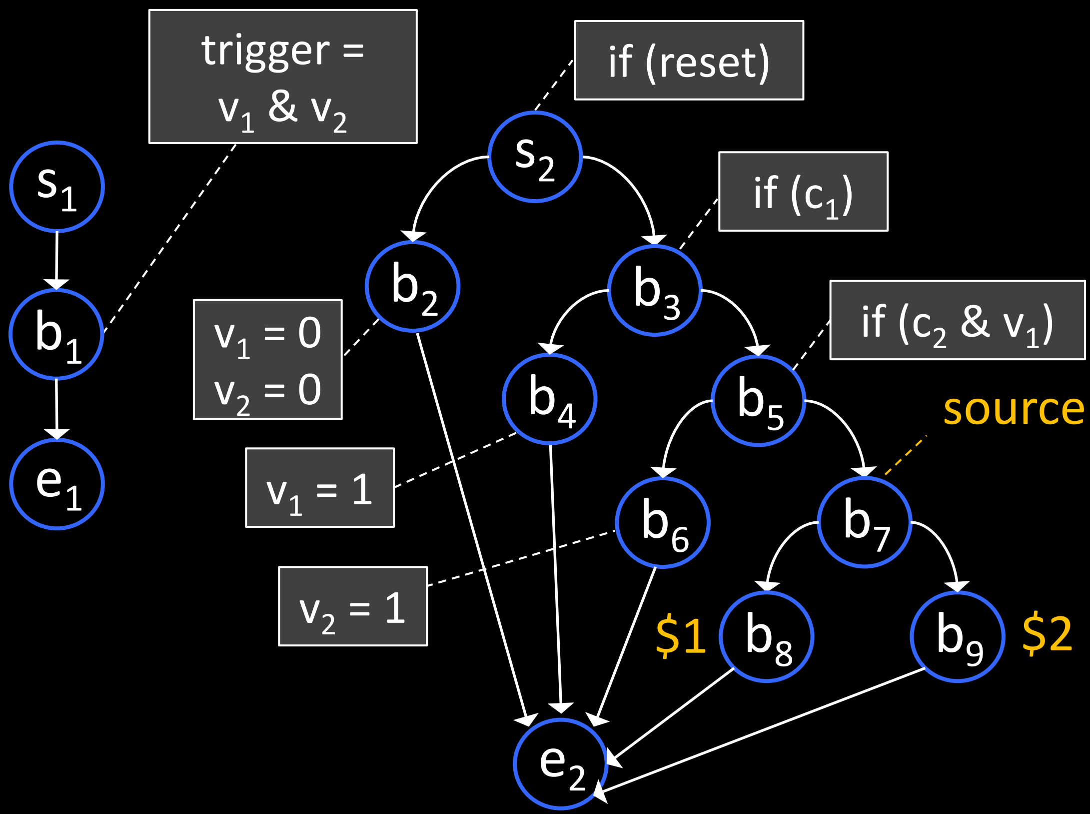

Extension directives:

1. parametrizable 1
2. bound-number 10
3. add-basic-blocks 2
4. add-edge ($b_7$, \$1)
5. add-edge ($b_7$, \$2)
6. add-edge (\$1, $e_2$)
7. add-edge (\$2, $e_2$)
8. drop-edge ($b_7$, $e_2$)

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**



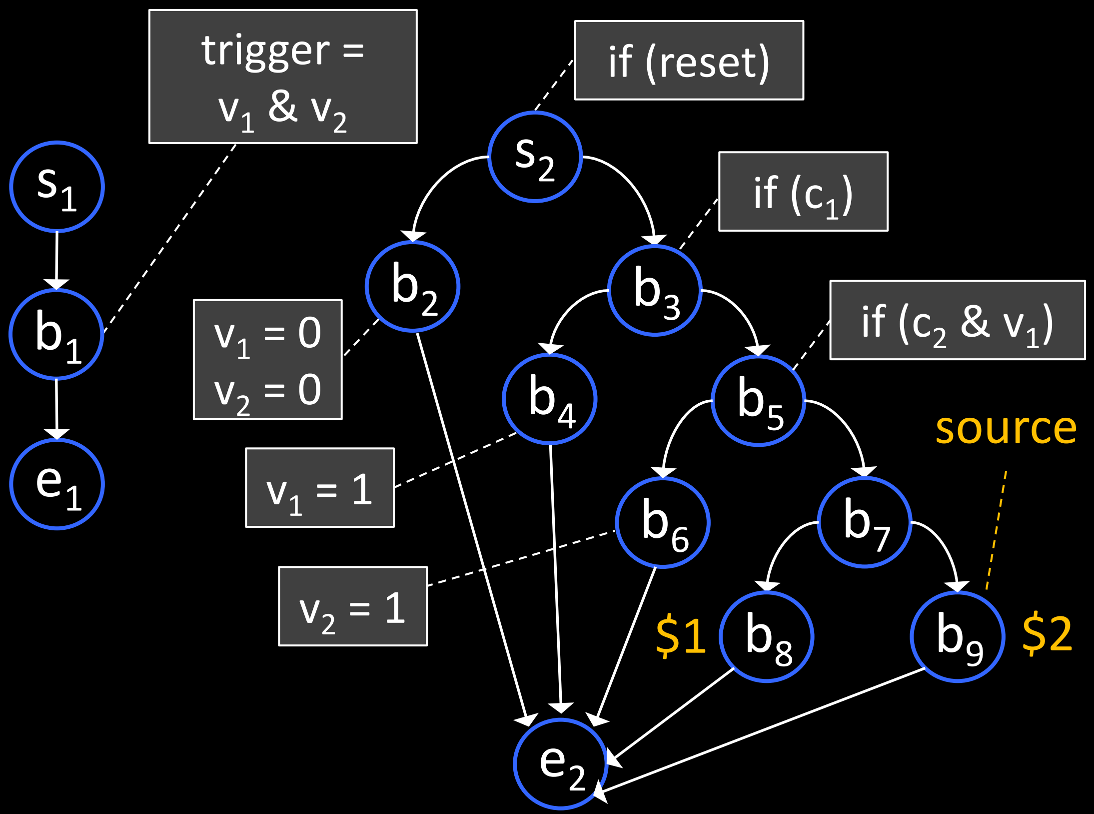

Extension directives:

1. parametrizable 1
2. bound-number 10
3. add-basic-blocks 2
4. add-edge ($b_7$, \$1)
5. add-edge ($b_7$, \$2)
6. add-edge (\$1, $e_2$)
7. add-edge (\$2, $e_2$)
8. drop-edge ($b_7$, $e_2$)
9. old-source-block $b_7$

# Hardware Trojan Library

## Handling Camouflaged Variants

- We need an automatic way to extend such basic implementations to find **camouflaged variants**




Extension directives:

1. parametrizable 1
2. bound-number 10
3. add-basic-blocks 2
4. add-edge ($b_7$, $1)
5. add-edge ($b_7$, $2)
6. add-edge ($1, $e_2$)
7. add-edge ($2, $e_2$)
8. drop-edge ($b_7$, $e_2$)
9. old-source-block $b_7$
10. up-source-block $2

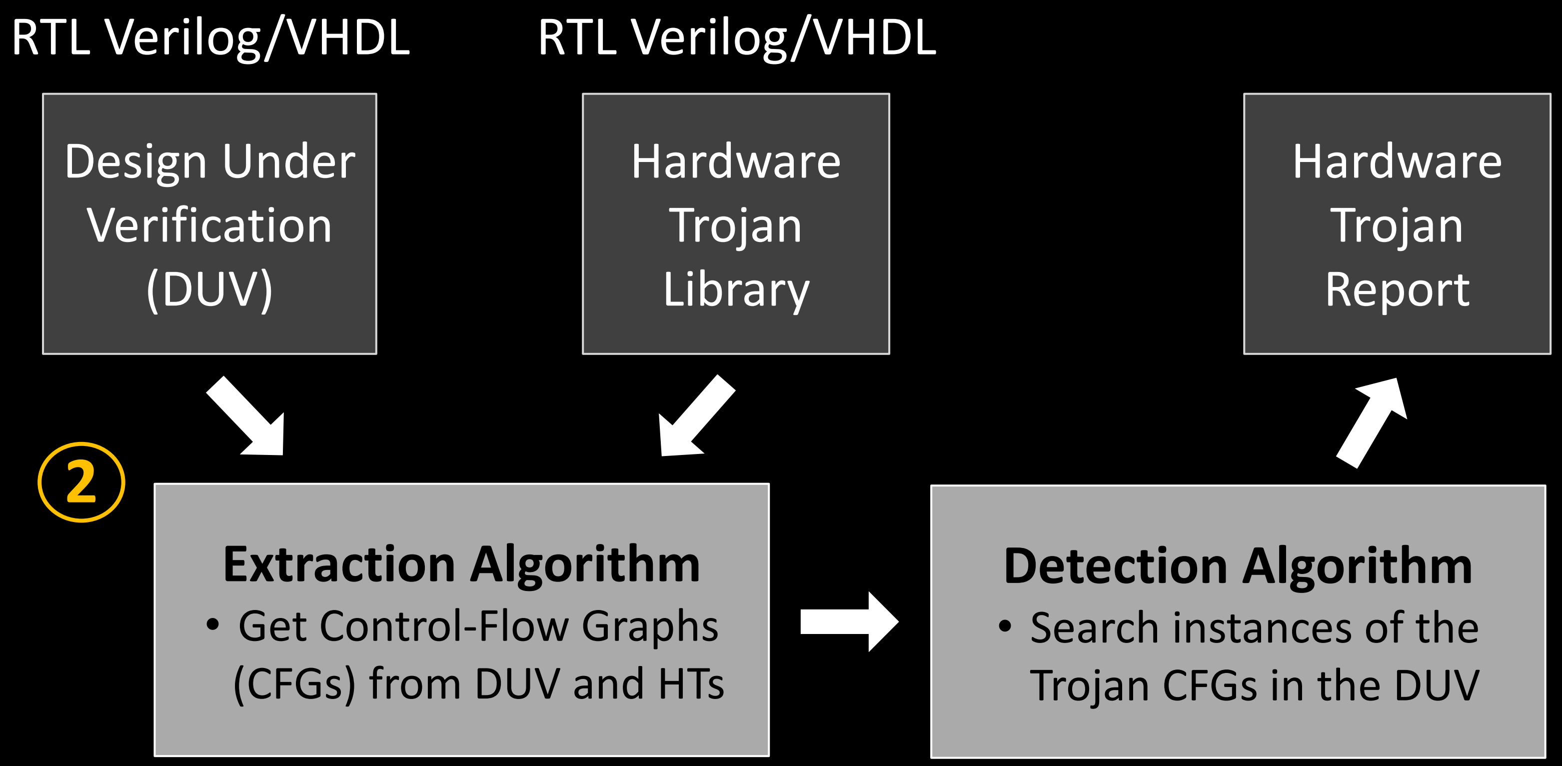
# Hardware Trojan Library

## Pros and Cons

- We defined a **Hardware Trojan (HT) Library** that includes the RTL implementations of known HT triggers and their camouflaged variants

| Pros | Cons |
| --- | --- |
| • Unique verification approach | • Unique verification approach |
| • Easy to extend the approach for new hardware Trojans | • Need of the implementations of the hardware Trojans |
| • Easy to customize the library to the needs of the user | • Only the hardware Trojans in the library or their variations can be detected |

# Hardware Trojan Detection
## Extraction Algorithm



# Hardware Trojan Detection

## Extraction Algorithm

```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```

# Hardware Trojan Detection

## Extraction Algorithm

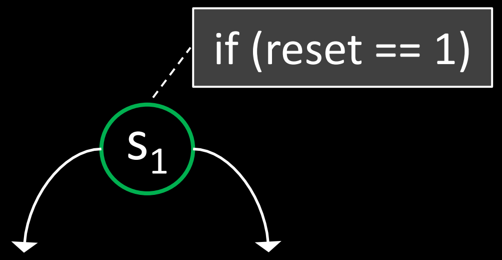
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```

$s_1$

# Hardware Trojan Detection

## Extraction Algorithm

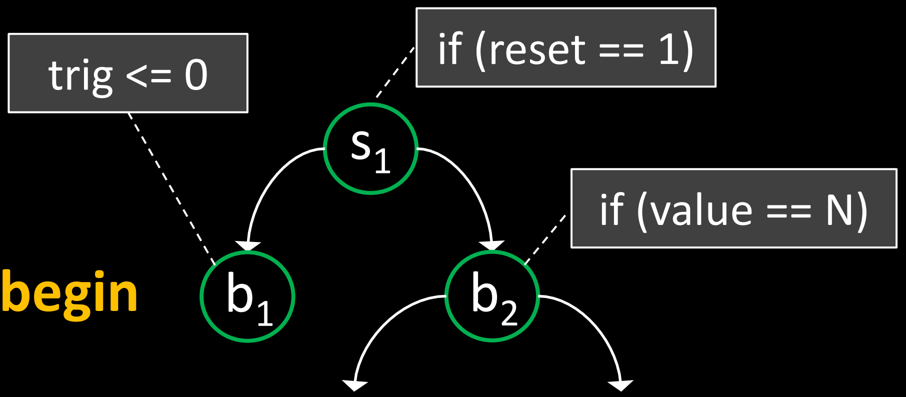
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



# Hardware Trojan Detection

## Extraction Algorithm

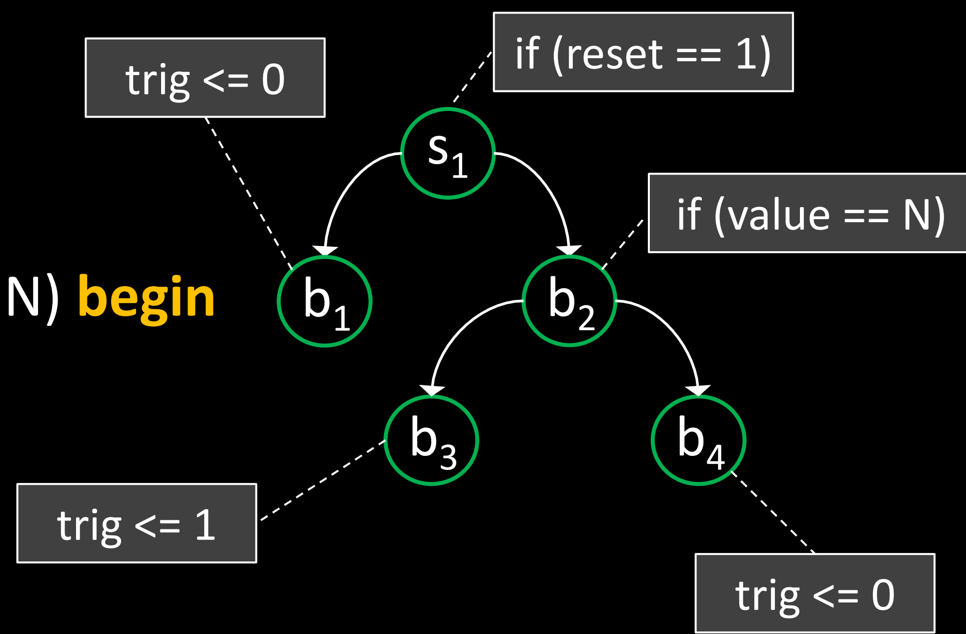
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128’hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



# Hardware Trojan Detection

## Extraction Algorithm

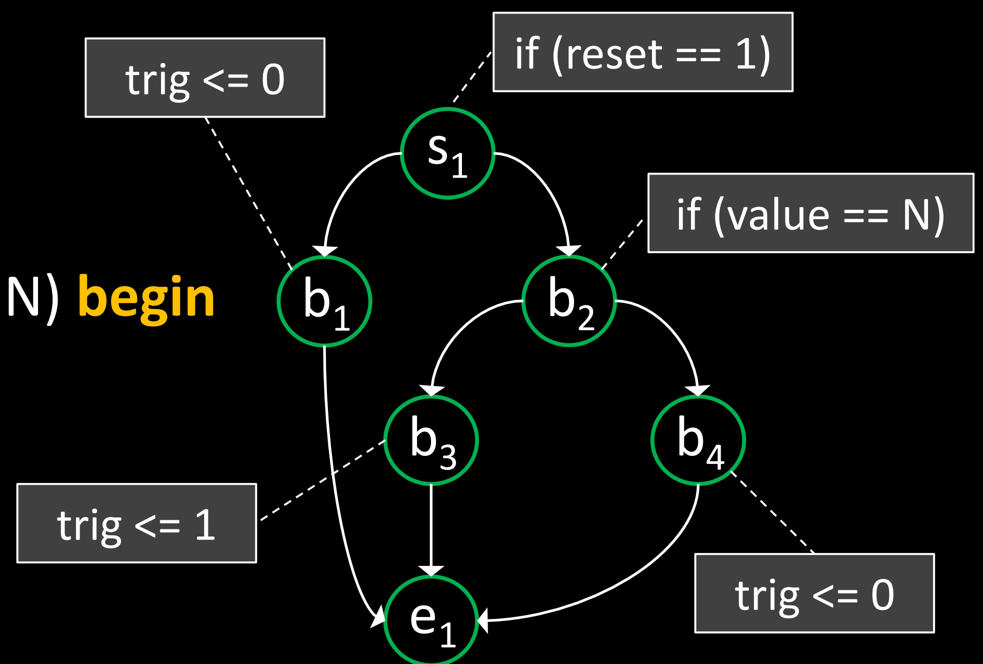
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



# Hardware Trojan Detection

## Extraction Algorithm

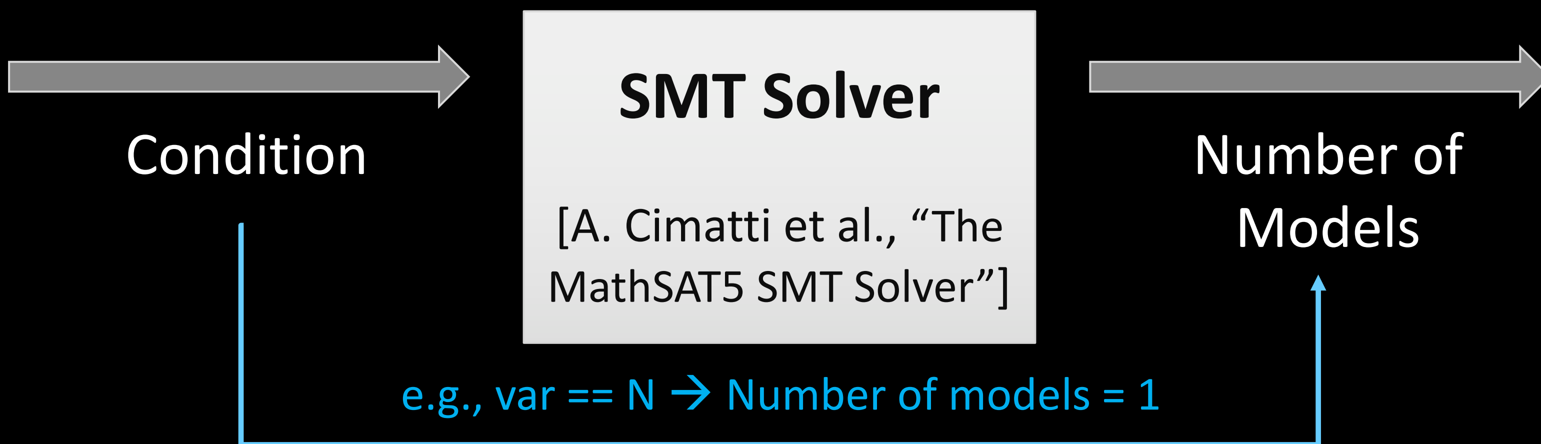
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128’hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



# Hardware Trojan Detection

## Extraction Algorithm: Probabilities

- To calculate the probabilities associated with the arcs, we use an approach based on a SMT solver

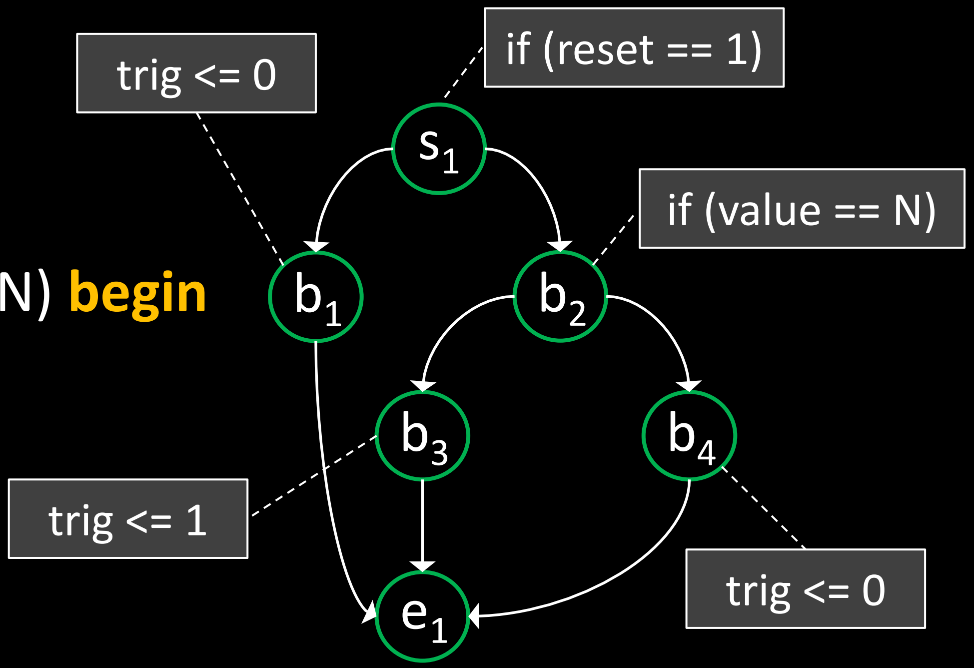



- Scalability? YES, conditions are simple enough!
  - Plus, simple conditions are short-circuited

# Hardware Trojan Detection

## Extraction Algorithm: Probabilities

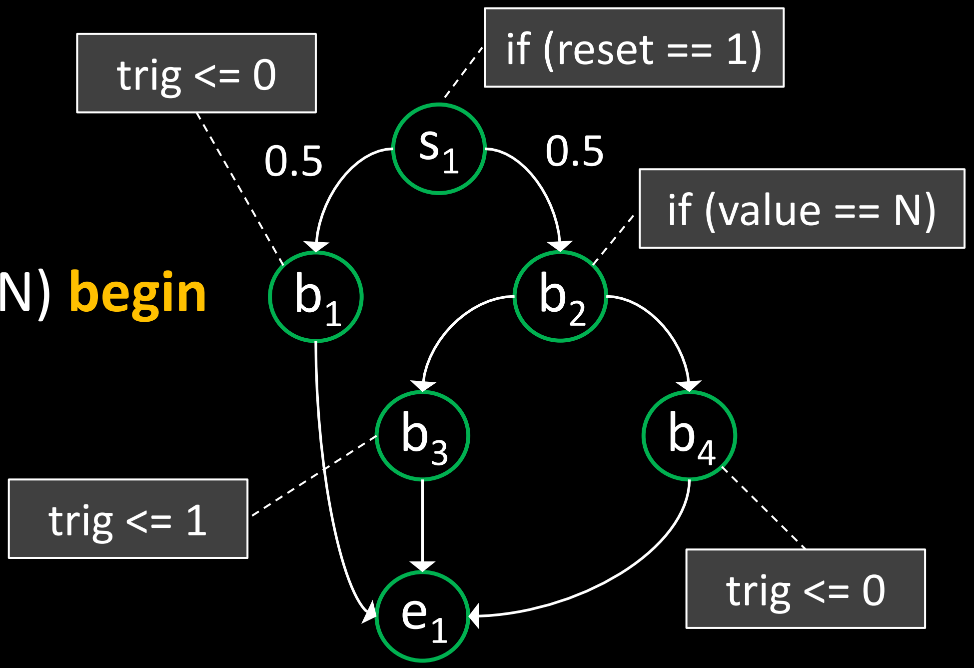
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



# Hardware Trojan Detection

## Extraction Algorithm: Probabilities

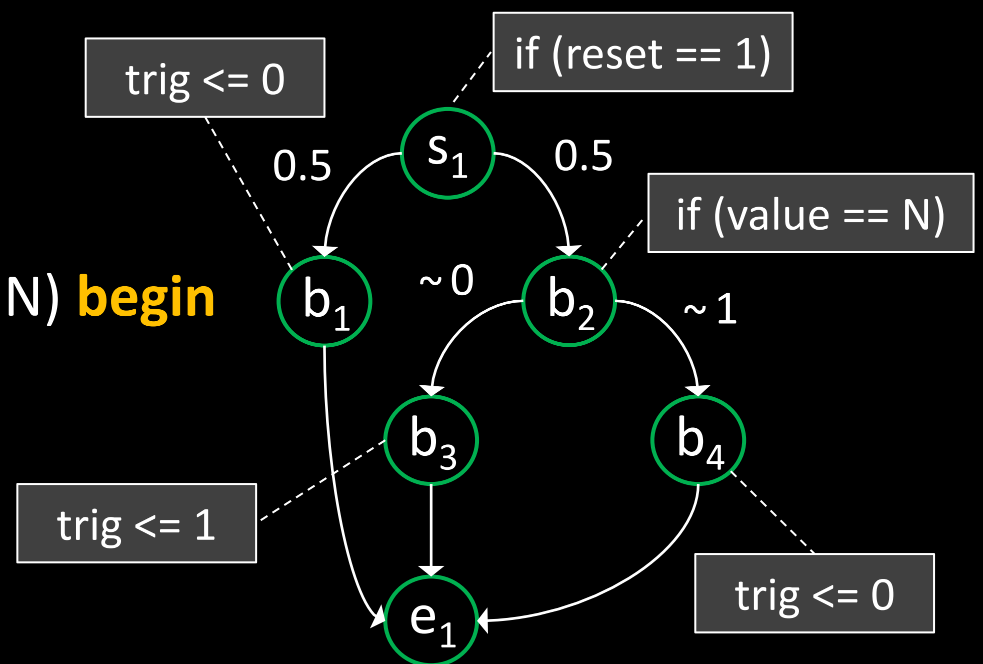
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
      if (reset == 1) begin
            trig <= 0;
      end else if (value == N) begin
            trig <= 1;
      end else begin
            trig <= 0;
      end
end
```



# Hardware Trojan Detection

## Extraction Algorithm: Probabilities

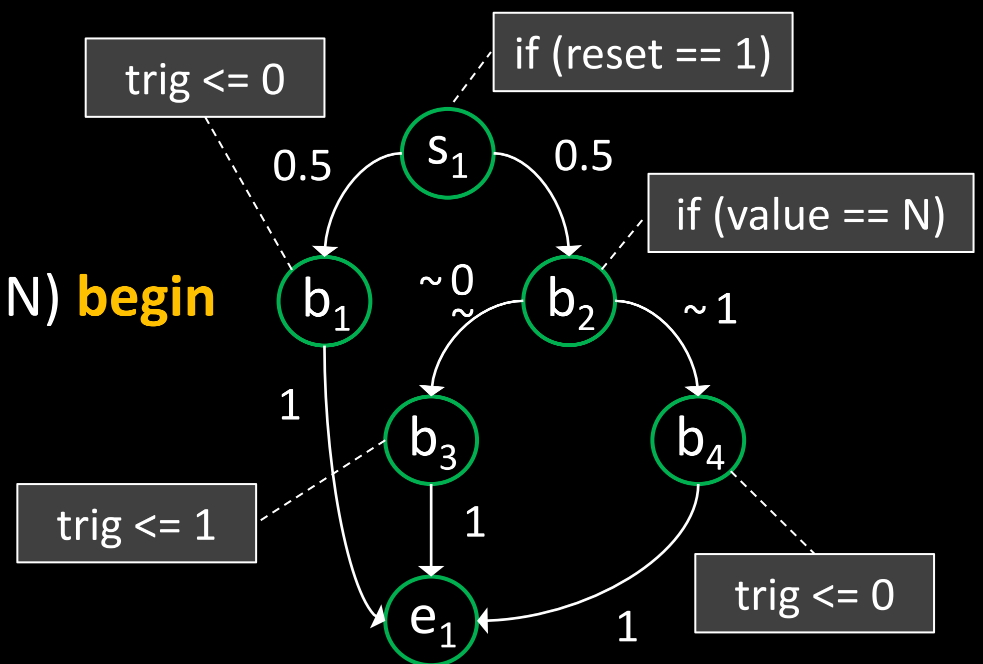
```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



# Hardware Trojan Detection

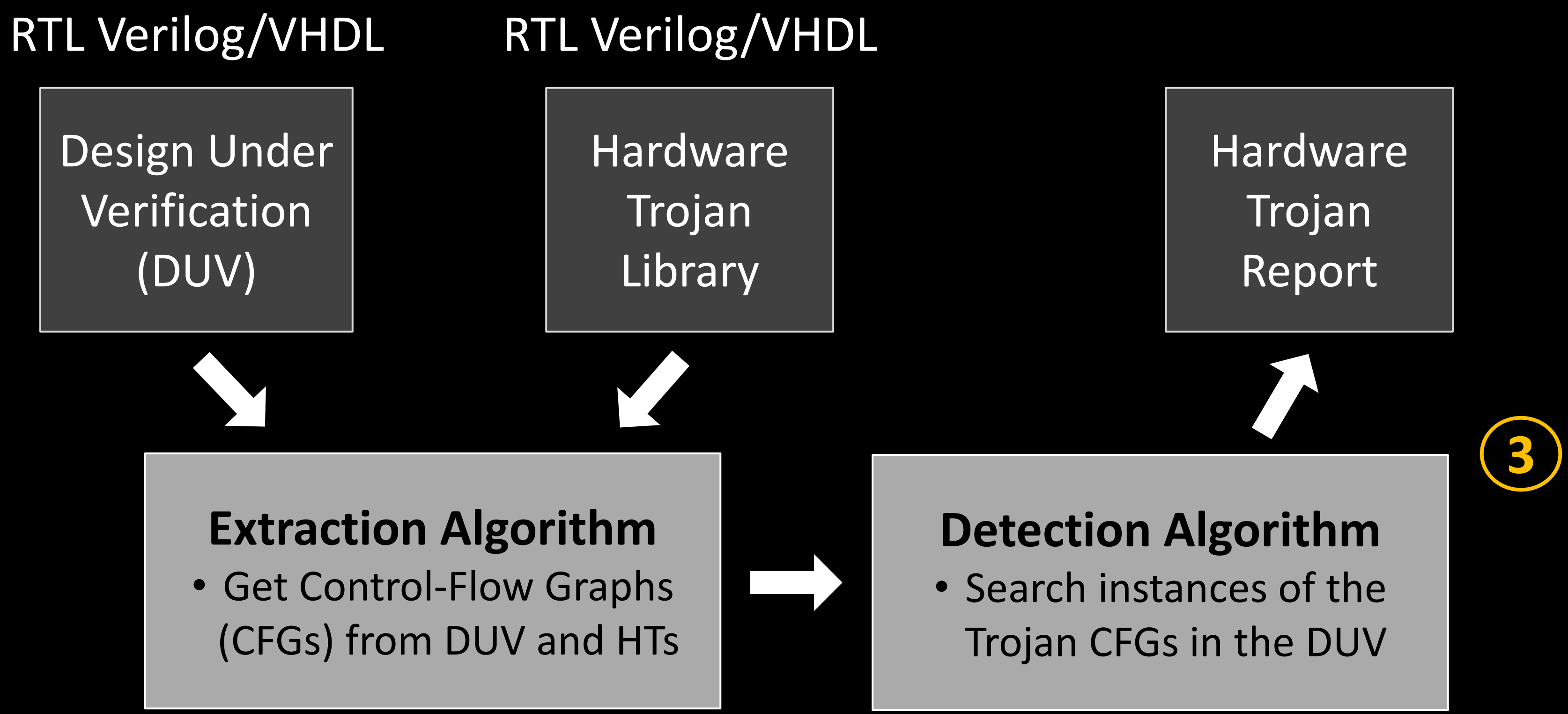
## Extraction Algorithm: Probabilities

```
module Trigger (input reset, input [127:0] value, output trig);

parameter N = 128'hffff_ffff_...._ffff;

always @(reset, value)
begin
    if (reset == 1) begin
        trig <= 0;
    end else if (value == N) begin
        trig <= 1;
    end else begin
        trig <= 0;
    end
end
```



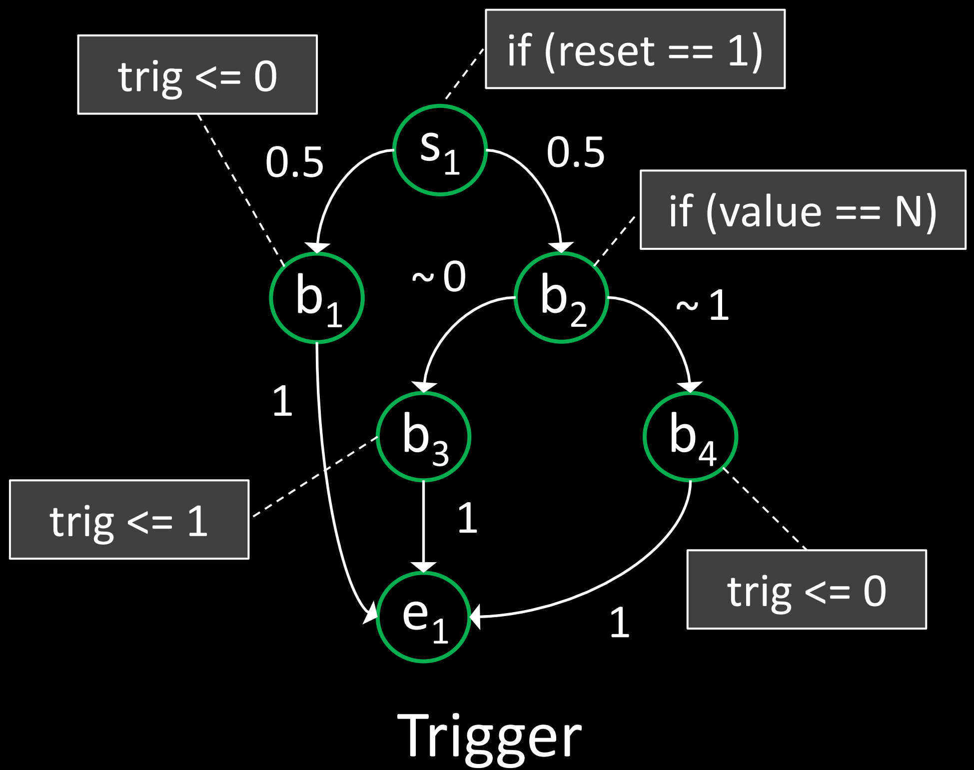
# Hardware Trojan Detection

## Detection Algorithm



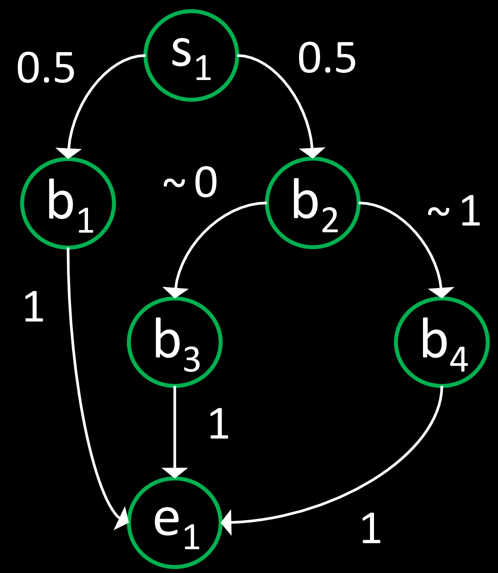
# Hardware Trojan Detection

## Detection Algorithm



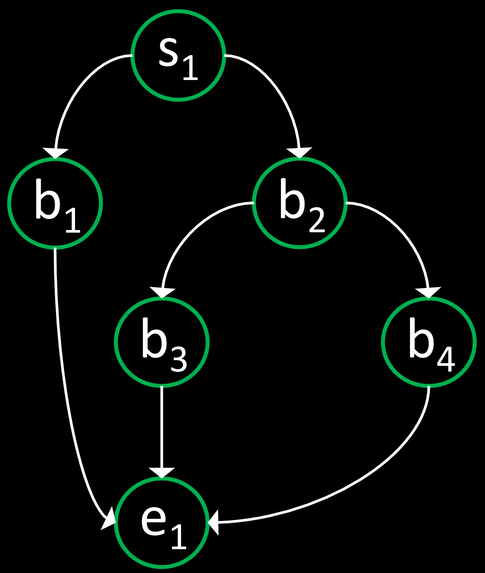
# Hardware Trojan Detection

## Detection Algorithm




Trigger

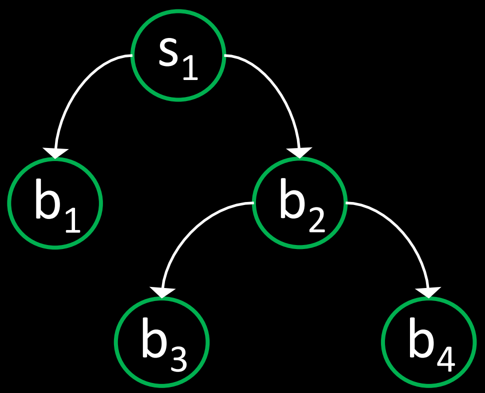
# Hardware Trojan Detection
## Detection Algorithm




Trigger

# Hardware Trojan Detection

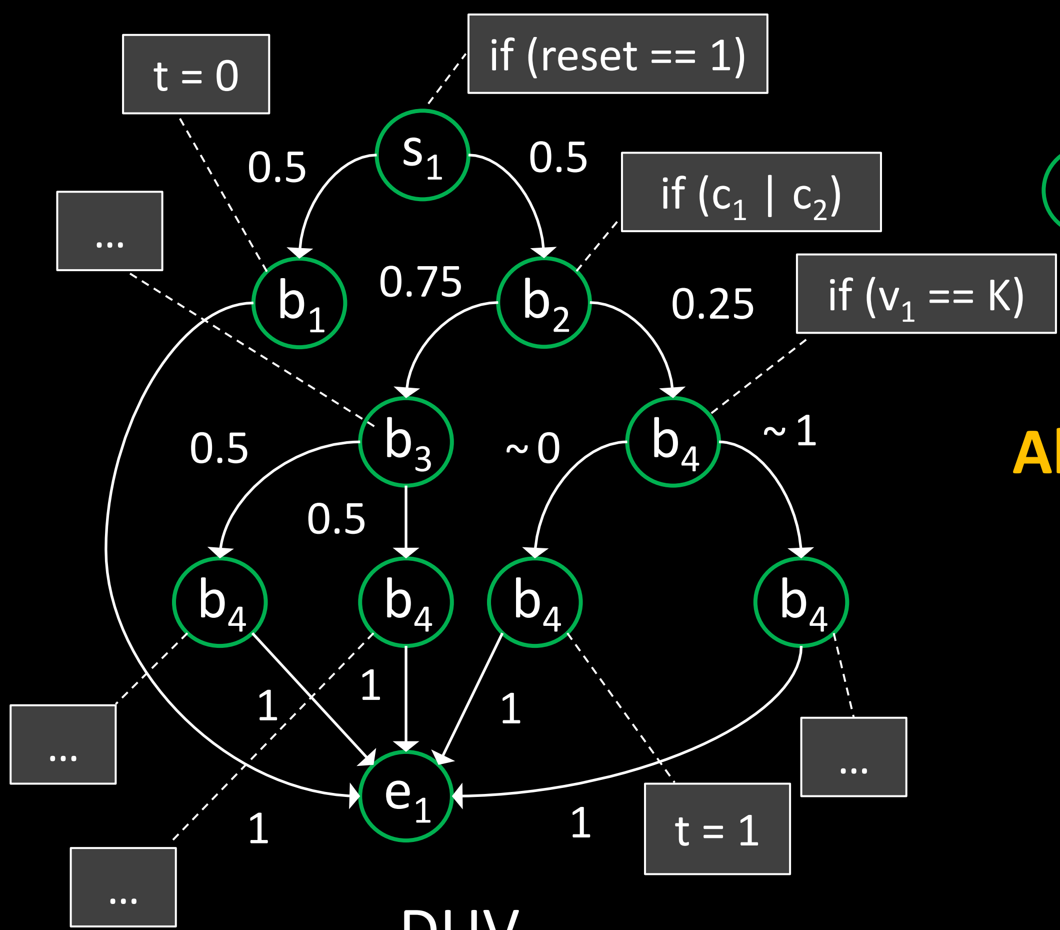
## Detection Algorithm



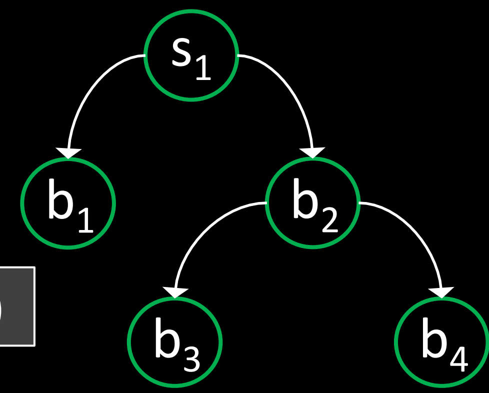

**Abstracted** Trigger

# Hardware Trojan Detection

## Detection Algorithm



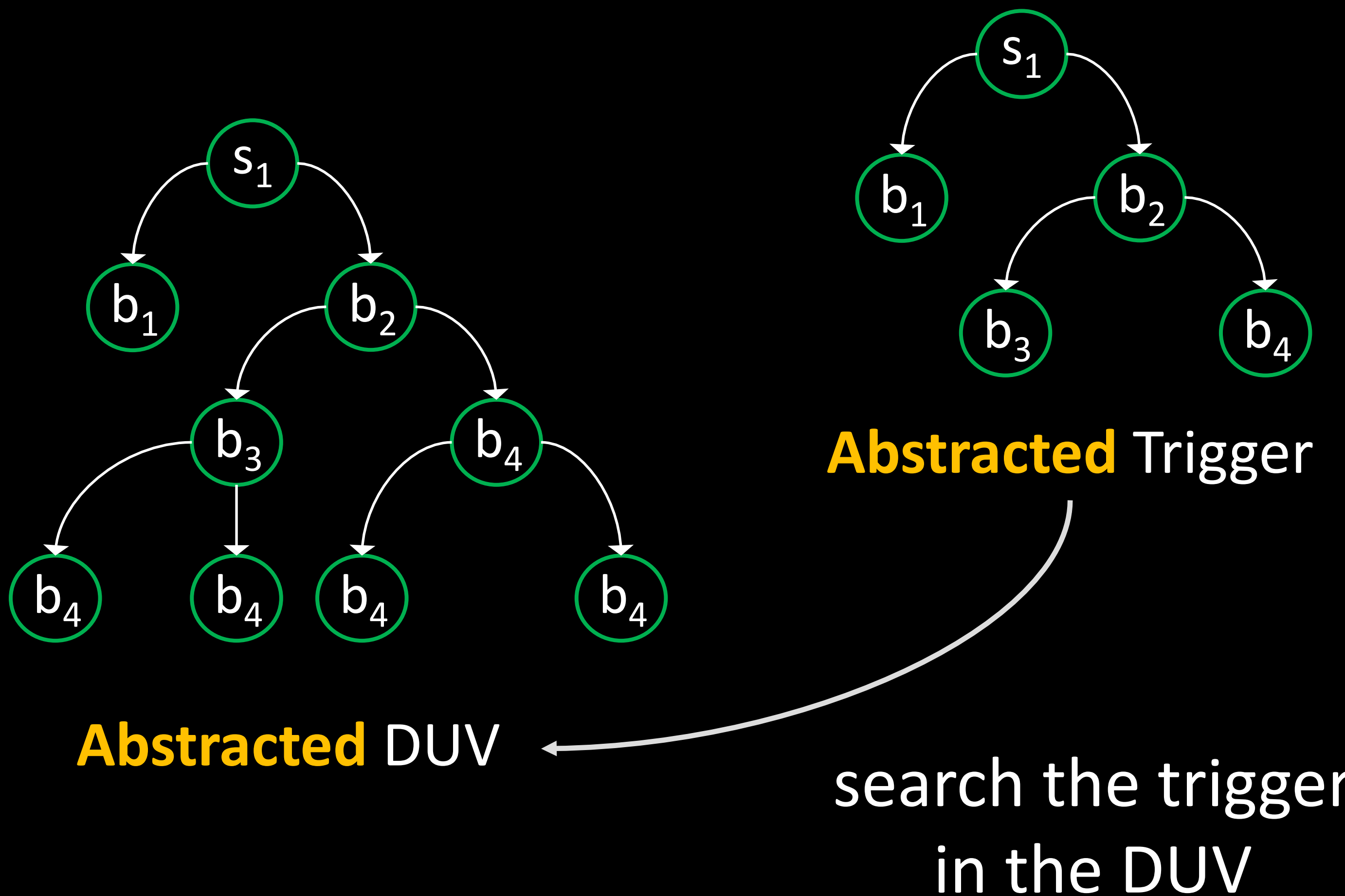

DUV




**Abstracted** Trigger

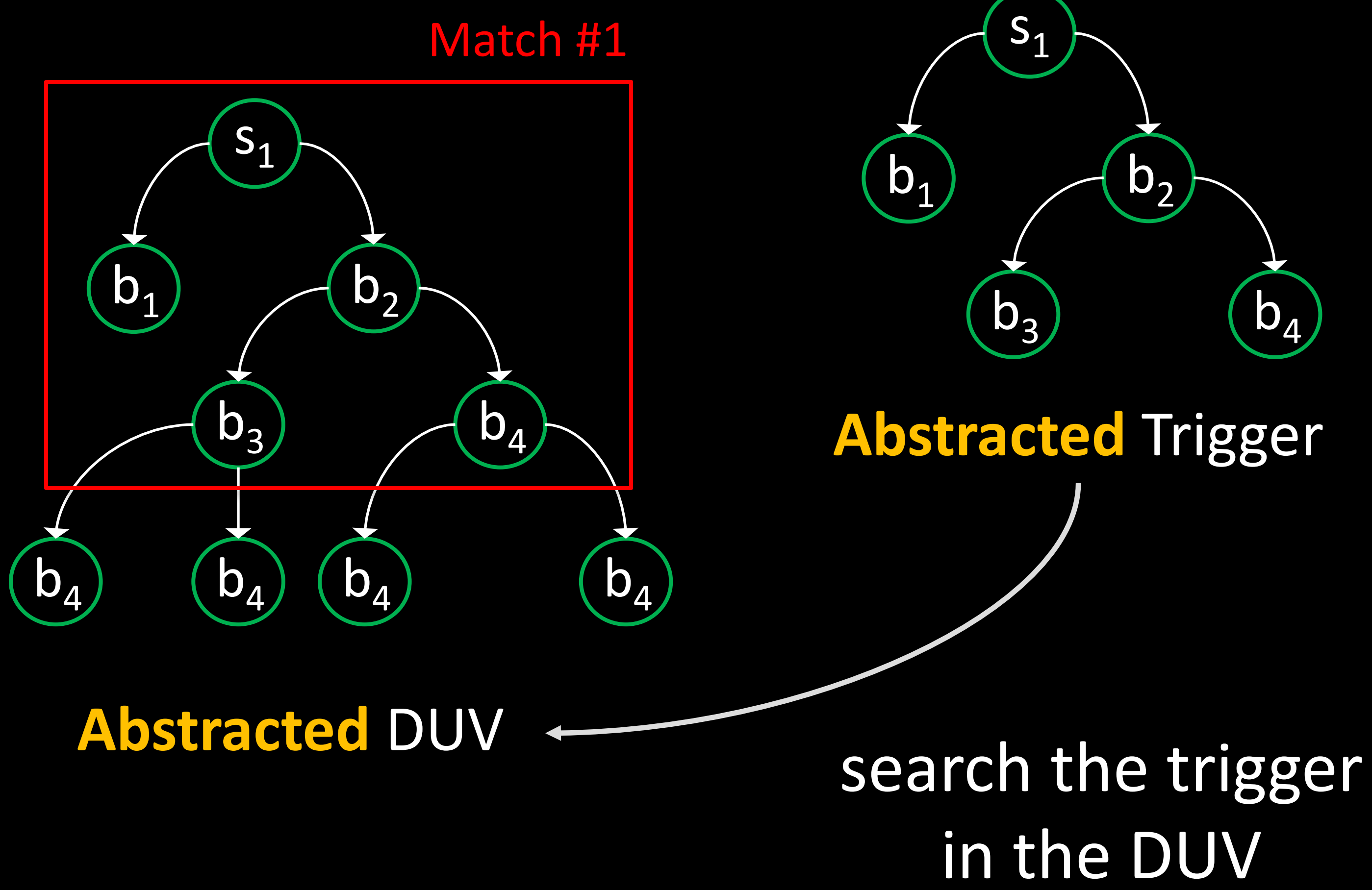
# Hardware Trojan Detection

## Detection Algorithm



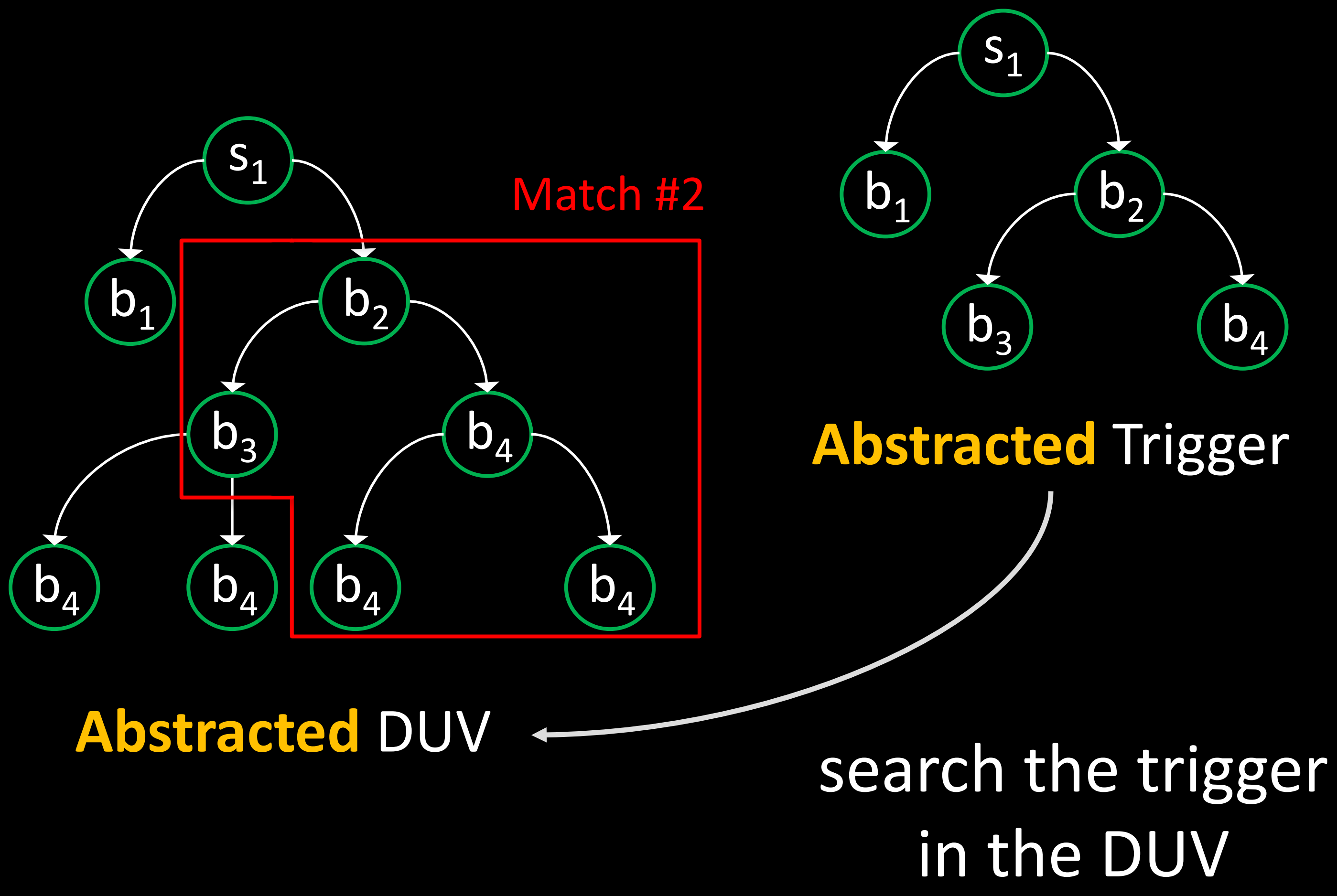
# Hardware Trojan Detection

## Detection Algorithm



# Hardware Trojan Detection

## Detection Algorithm



# Hardware Trojan Detection

## Detection Algorithm: Confidence

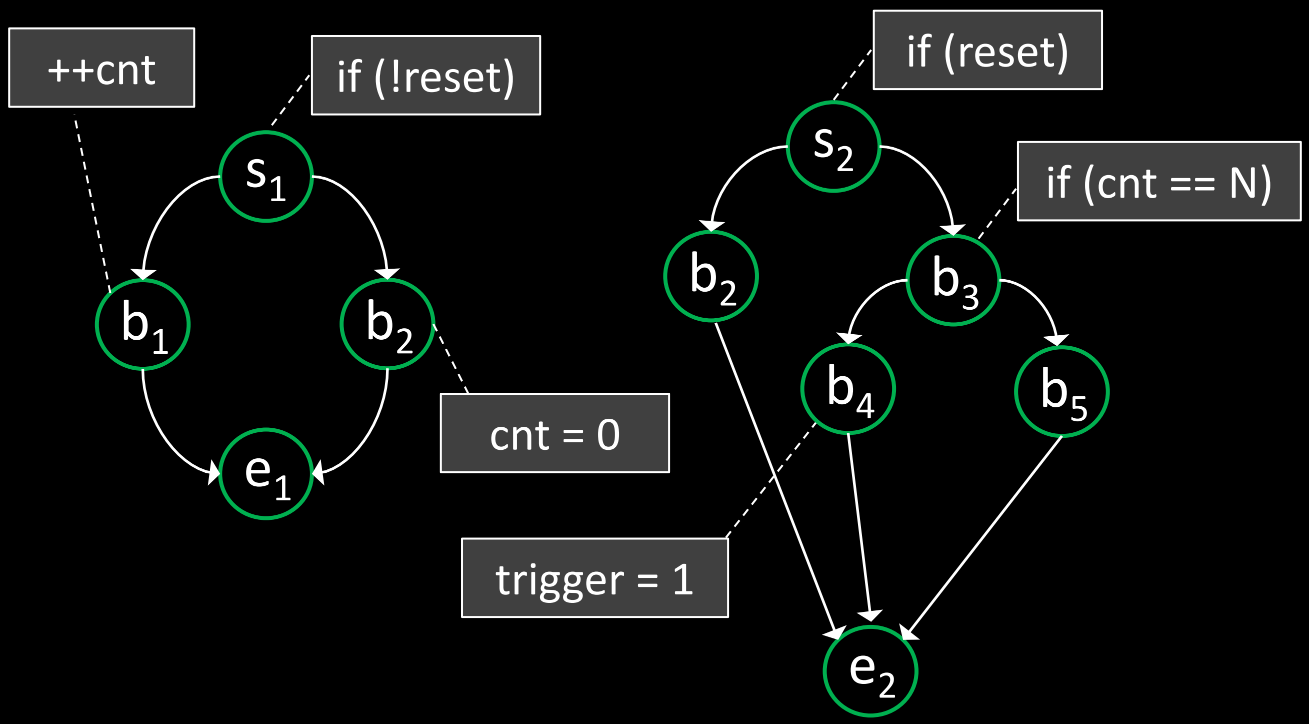
- Some Hardware Trojans can be similar to actual legal code: we need to give a confidence value for each match returned by the detection alg.
  - The confidence value is in the range [0, 1]
  - 1 → highest confidence that is a Trojan
- For each match we evaluate 4 conditions $c_1$, $c_2$, $c_3$ and $c_4$ → confidence is a linear combination of those conditions (weights vary with triggers)

# Hardware Trojan Detection

## Detection Algorithm: Confidence

$c_1$: presence of variables with known behavior

Trigger in the HT Library

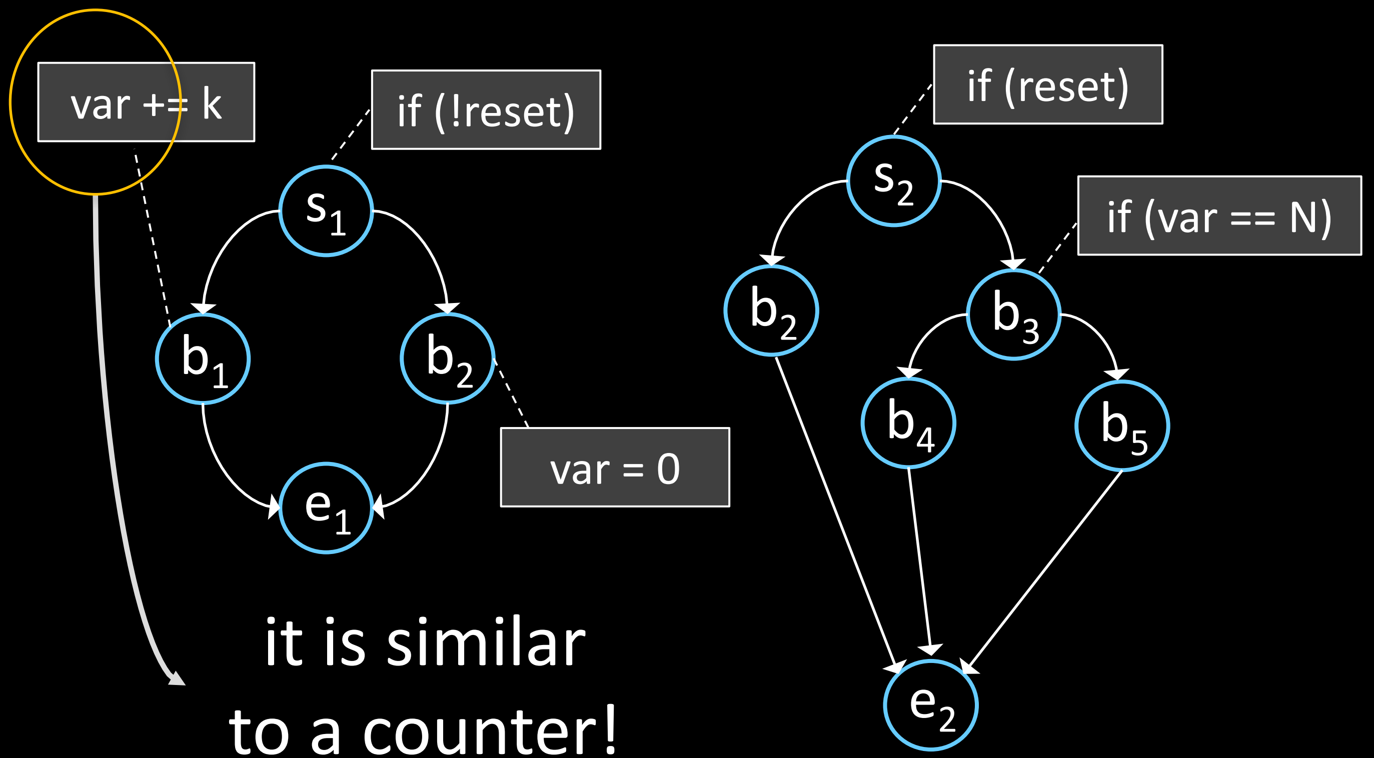


# Hardware Trojan Detection

## Detection Algorithm: Confidence

$c_1$: presence of variables with known behavior

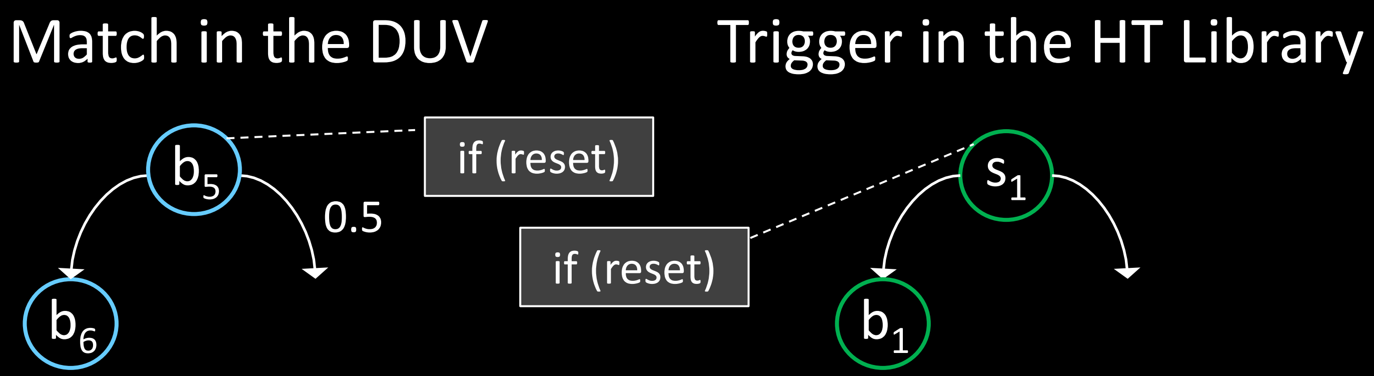
Match in the DUV



# Hardware Trojan Detection

## Detection Algorithm: Confidence

$c_2$: presence of suspicious reset logics

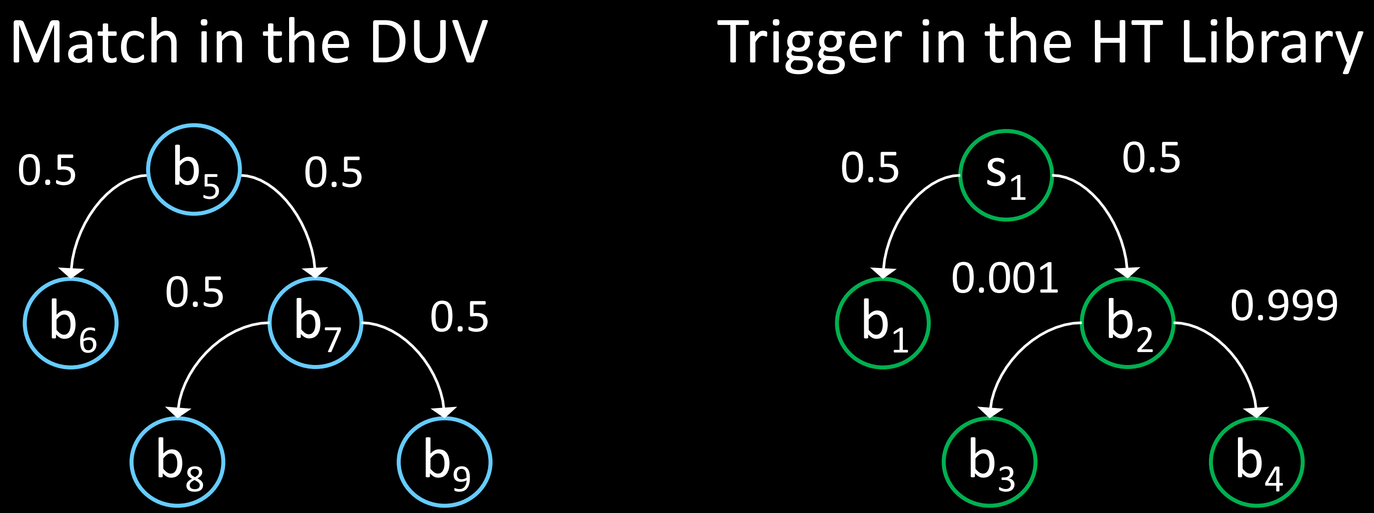



- Same reset mechanism of the process?
- Suspicious variables are reset?

# Hardware Trojan Detection

## Detection Algorithm: Confidence

$c_3$: average distance of the probabilities



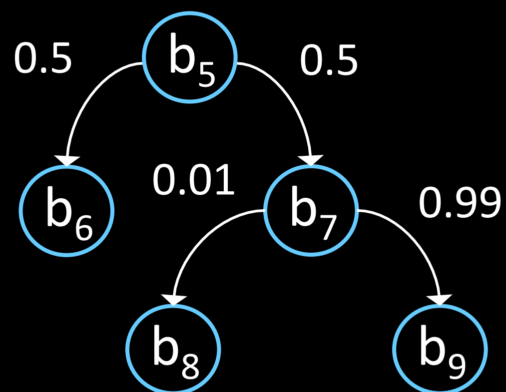

**confidence =** $1 - [\,|0.5 - 0.5| + |0.5 - 0.5| + |0.5 - 0.001| + |0.5 - 0.999|\,] = 0.002$

# Hardware Trojan Detection

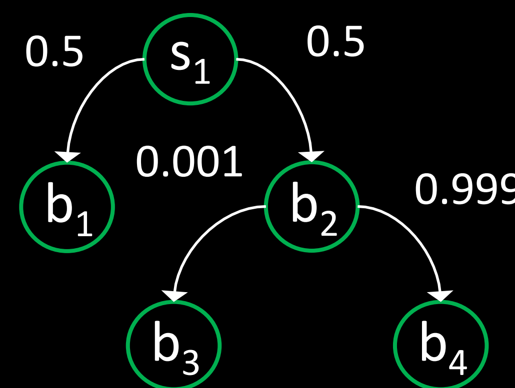
## Detection Algorithm: Confidence

$c_3$: average distance of the probabilities

Match in the DUV




Trigger in the HT Library




$$\textbf{confidence} = 1 - [\,|0.5 - 0.5| + |0.5 - 0.5| + |0.01 - 0.001| + |0.99 - 0.999|\,] = 0.892$$

# Hardware Trojan Detection

## Detection Algorithm: Confidence

$c_4$: is there a payload that is affine to the trigger?

RTL Verilog/VHDL

Hardware Trojan Library → Added known implementations of **HT payloads**

- The payloads are searched as well in the DUV
- Are there a matched payload and matched trigger that share some variables?

# Experimental Results

- We verified the effectiveness of our approach by considering the Trust-HUB Benchmarks and the Cryptoplatform (component from OpenCores)
- We created a HT Library that includes the same types of HTs (but not the same code) of the HTs that have been included in the benchmarks
- The goal here is to show that our verification approach can help users to distinguish HTs

# Experimental Results

## HT Library (Triggers)

| | Cheat codes | |
|---|---|---|
| Name | Blocks | Edges |
| Cheat-T001 | 4 | 4 |
| Cheat-T002 | 5 | 6 |
| Cheat-T003 | 6 | 7 |
| Cheat-T004 | 16 | 21 |
| Cheat-T005 | 11 | 14 |
| Cheat-T006 | 11 | 14 |

| | Timebombs | |
|---|---|---|
| Name | Blocks | Edges |
| Time-T001 | 13 | 16 |
| Time-T002 | 14 | 19 |
| Time-T003 | 12 | 15 |
| Time-T004 | 6 | 7 |
| Time-T005 | 14 | 17 |

| | Dead machines | |
|---|---|---|
| Name | Blocks | Edges |
| Mach-T001 | 10 | 11 |
| Mach-T002 | 11 | 13 |

# Experimental Results

## HT Library (Payloads)

| | Payloads | | |
|---|---|---|---|
| Name | Effect | Blocks | Edges |
| Payload-T001 | Infor. leakage | 16 | 21 |
| Payload-T002 | Increase Power | 8 | 9 |
| Payload-T003 | Covert Channel | 10 | 13 |
| Payload-T004 | Leakage Current | 12 | 15 |
| Payload-T005 | Modify memory | 7 | 7 |
| Payload-T006 | Modify output | 7 | 7 |

# Experimental Results

## Characteristics of Benchmarks

| | Trust-HUB Benchmarks | | | | |
|---|---|---|---|---|---|
| Name | # Diff. Instances | Min. # Blocks | Max. # Blocks | Min. # Edges | Max. # Edges |
| AES | 16 | 2101 | 2150 | 3160 | 3236 |
| RS232 | 10 | 130 | 159 | 184 | 233 |
| BasicRSA | 4 | 81 | 93 | 119 | 139 |

| | Cryptoplatform (CPU + memory + 5 crypto cores) | | | | |
|---|---|---|---|---|---|
| Name | # Diff. Instances | Min. # Blocks | Max. # Blocks | Min. # Edges | Max. # Edges |
| Crypto | 6 | 4402 | 4424 | 6503 | 6537 |

# Experimental Results

## Quantitative Evaluation

| | Trust-HUB Benchmarks | | | | |
|---|---|---|---|---|---|
| Family | [A] | [B] | [C] | [C]* | **This work** |
| *AES* | 3/18 | 9/18 | 0/18 | 18/18 | 18/18 |
| *RS232* | 0/10 | 0/10 | 9/10 | 10/10 | 10/10 |
| *BasicRSA* | 0/4 | 2/4 | 4/4 | 4/4 | 4/4 |

**A →** [J. Rajendran et al., "Detecting Malicious Modifications of Data in Third-Party Intellectual Property Cores", DAC '15]

**B →** [J. Rajendran et al., "Formal Security Verification of Third-Party Intellectual Property Cores for Information Leakage", VLSID '16]

**C →** [S. K. Haider et al., "HaTCh: Hardware Trojan Catcher", '14]

* Assuming they are activated during the learning phase

# Experimental Results

## Qualitative Evaluation

| | Proposed Approach for Trust-HUB Benchmarks | | | | |
|---|---|---|---|---|---|
| Name | Matches | $Conf_{HT}$ | $Conf_{MAX}$ | False+ | Time (s) |
| AES-T800 | 9 | 0.93 | 0.65 | 0 | 5.04 |
| AES-T1400 | 81 | 0.99 | 0.69 | 0 | 4.85 |
| AES-T1900 | 11 | 0.97 | 0.72 | 0 | 4.82 |
| RS232-T100 | 7 | 0.36 | 0.50 | 2 | 4.12 |
| BasicRSA-T100 | 4 | 0.25 | 0.25 | 3 | 1.13 |

(Full results in the paper or in the poster)

# Experimental Results

## Qualitative Evaluation

| | Proposed Approach for Cryptoplatform | | | | |
|---|---|---|---|---|---|
| Name | Matches | $Conf_{HT}$ | $Conf_{MAX}$ | False+ | Time (s) |
| Crypto-T000 | 23 | N/A | 0.35 | N/A | 11.80 |
| Crypto-T100 | 34 | 0.81 | 0.39 | 0 | 12.88 |
| - | 34 | 0.72 | 0.39 | 0 | 12.88 |
| Crypto-T200 | 31 | 0.96 | 0.71 | 0 | 13.43 |
| Crypto-T300 | 42 | 0.88 | 0.29 | 0 | 15.03 |
| Crypto-T400 | 34 | 0.90 | 0.50 | 0 | 15.67 |

# Conclusions

- We presented an automatic approach for the detection of hardware Trojans at RTL

1. Our approach is **general**: it adopts an approach independent from the specific hardware Trojan
2. Our approach is **extendible**: new Trojans can be easily added to the Hardware Trojan Library
3. Our approach is **fast**: it takes only few seconds to find hardware Trojans in large DUVs

# Efficient Control-Flow Subgraph Matching for Detecting Hardware Trojans in RTL Models

## Questions?

**Speaker**: Luca Piccolboni
Columbia University, NY, USA
University of Verona, Verona, Italy