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.

Hardware Trojan

- **Trigger Logic** activates the malicious behavior under specific conditions.
- **Payload Logic** 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**

**Extraction Algorithm**
- Get Control-Flow Graphs (CFGs) from DUV and HTs

**Detection Algorithm**
- Search instances of the Trojan CFGs in the DUV
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

RTL Verilog/VHDL

Design Under Verification (DUV)

RTLS Verilog/VHDL

Hardware Trojan Library

Extraction Algorithm

• Get Control-Flow Graphs (CFGs) from DUV and HTs

RTL Verilog/VHDL

Hardware Trojan Report

Detection Algorithm

• Search instances of the Trojan CFGs in the DUV
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.

```plaintext
++cnt
if (reset)

s1

b1

e1

b2

if (reset)
cnt = 0

s2

b3

b4

if (reset)
cnt == N

b5

b6

if (reset)

e2

trigger = 1
```
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**
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 (b7, $1$)
5. add-edge (b7, $2$)
6. add-edge ($1$, e2)
7. add-edge ($2$, e2)
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 (b₇, $1$
5. add-edge (b₇, $2$
6. add-edge ($1$, e₂)
7. add-edge ($2$, e₂)
8. drop-edge (b₇, e₂)
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 (b₇, $1$
5. add-edge (b₇, $2$
6. add-edge ($1$, e₂)
7. add-edge ($2$, e₂)
8. drop-edge (b₇, e₂)
9. old-source-block b₇
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

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Unique verification approach</td>
<td>• Unique verification approach</td>
</tr>
<tr>
<td>• Easy to extend the approach for new hardware Trojans</td>
<td>• Need of the implementations of the hardware Trojans</td>
</tr>
<tr>
<td>• Easy to customize the library to the needs of the user</td>
<td>• Only the hardware Trojans in the library or their variations can be detected</td>
</tr>
</tbody>
</table>
Hardware Trojan Detection
Extraction Algorithm

RTL Verilog/VHDL

Design Under Verification (DUV)

RTL Verilog/VHDL

Hardware Trojan Library

RTL Verilog/VHDL

Hardware Trojan Report

Extraction Algorithm
• Get Control-Flow Graphs (CFGs) from DUV and HTs

Detection Algorithm
• Search instances of the Trojan CFGs in the DUV
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

```verilog
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
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
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

**RTL Verilog/VHDL**

1. **Design Under Verification (DUV)**
2. **Hardware Trojan Library**

**Extraction Algorithm**
- Get Control-Flow Graphs (CFGs) from DUV and HTs

**RTL Verilog/VHDL**

**Detection Algorithm**
- Search instances of the Trojan CFGs in the DUV

**Hardware Trojan Report**
Hardware Trojan Detection

Detection Algorithm

![State Transition Diagram]

- **State Variables:**
  - `s1`, `b1`, `b2`, `b3`, `b4`, `e1`, Trigger

- **Transitions:*
  - `trig <= 0`:
    - From `s1` to `b1`: `0.5`
    - From `b1` to `b4`: `1`
    - From `b4` to `e1`: `1`
  - `if (reset == 1)`:
    - From `s1` to `b1`: `0.5`
  - `if (value == N)`:
    - From `b2` to `s1`: `~0`
    - From `b3` to `b2`: `~1`

- **Initial State:**
  - `s1`

- **Final State:**
  - `e1`
Hardware Trojan Detection
Detection Algorithm

\begin{tikzpicture}
    \node [circle, draw, fill=green!50] (s1) at (0,0) {$s_1$};
    \node [circle, draw, fill=green!50] (b1) at (-1,-1) {$b_1$};
    \node [circle, draw, fill=green!50] (b2) at (-1,-2) {$b_2$};
    \node [circle, draw, fill=green!50] (b3) at (-2,-3) {$b_3$};
    \node [circle, draw, fill=green!50] (b4) at (-2,-4) {$b_4$};
    \node [circle, draw, fill=green!50] (e1) at (-3,-5) {$e_1$};
    \path [->] (s1) edge node [above] {0.5} (b1);
    \path [->] (s1) edge node [above] {0.5} (b2);
    \path [->] (b1) edge node [above] {$\sim 0$} (b2);
    \path [->] (b2) edge node [right] {$\sim 1$} (b4);
    \path [->] (b3) edge node [right] {1} (b4);
    \path [->] (b4) edge node [right] {1} (b3);
    \path [->] (e1) edge node [right] {1} (b3);
    \path [->] (e1) edge node [right] {1} (b4);
\end{tikzpicture}
Hardware Trojan Detection
Detection Algorithm

![Diagram](image)
Hardware Trojan Detection

Detection Algorithm

Abstracted Trigger
Hardware Trojan Detection
Detection Algorithm

\[ t = 0 \]
\[ \text{if (reset == 1)} \]
\[ s_1 \]
\[ b_1 \]
\[ 0.5 \]
\[ b_2 \]
\[ 0.75 \]
\[ b_3 \]
\[ 0.5 \]
\[ b_4 \]
\[ 0 \]
\[ b_4 \]
\[ \sim 0 \]
\[ b_4 \]
\[ \sim 1 \]
\[ e_1 \]
\[ 1 \]
\[ 1 \]
\[ 1 \]
\[ \text{DUV} \]

\[ 1 \]
\[ t = 1 \]

\[ \text{Abstracted Trigger} \]
Hardware Trojan Detection

Detection Algorithm

Abstracted DUV

Abstracted Trigger

search the trigger in the DUV
Hardware Trojan Detection

Detection Algorithm

Abstracted DUV

Match #1

search the trigger in the DUV

Abstracted Trigger
Hardware Trojan Detection

Detection Algorithm

Abstracted DUV

Match #2

Abstracted Trigger

search the trigger in the DUV
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

\[
\begin{align*}
\text{if (reset)} & : \\
\text{var} & = 0 \\
\text{if (!reset)} & : \\
\text{var} & += k
\end{align*}
\]

it is similar to a counter!
Hardware Trojan Detection
Detection Algorithm: Confidence

c₂: presence of suspicious reset logics

- Match in the DUV
- Trigger in the HT Library

- Same reset mechanism of the process?
- Suspicious variables are reset?
Hardware Trojan Detection

Detection Algorithm: Confidence

c₃: average distance of the probabilities

Match in the DUV

Trigger in the HT Library

\[
\text{confidence} = 1 - \left[ |0.5 - 0.5| + |0.5 - 0.5| + |0.5 - 0.001| + |0.5 - 0.999| \right] = 0.002
\]
Hardware Trojan Detection
Detection Algorithm: Confidence

c_3: average distance of the probabilities

Match in the DUV

Match in the DUV

Trigger in the HT Library

**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

c₄: is there a payload that is affine to the trigger?

• 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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Blocks</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheat-T001</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cheat-T002</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Cheat-T003</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Cheat-T004</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Cheat-T005</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Cheat-T006</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Blocks</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-T001</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>Time-T002</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>Time-T003</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>Time-T004</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Time-T005</td>
<td>14</td>
<td>17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Blocks</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach-T001</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Mach-T002</td>
<td>11</td>
<td>13</td>
</tr>
</tbody>
</table>
## Experimental Results

### HT Library (Payloads)

<table>
<thead>
<tr>
<th>Name</th>
<th>Effect</th>
<th>Blocks</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payload-T001</td>
<td>Infor. leakage</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>Payload-T002</td>
<td>Increase Power</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Payload-T003</td>
<td>Covert Channel</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>Payload-T004</td>
<td>Leakage Current</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>Payload-T005</td>
<td>Modify memory</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Payload-T006</td>
<td>Modify output</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
## Experimental Results

### Characteristics of Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th># Diff. Instances</th>
<th>Min. # Blocks</th>
<th>Max. # Blocks</th>
<th>Min. # Edges</th>
<th>Max. # Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>16</td>
<td>2101</td>
<td>2150</td>
<td>3160</td>
<td>3236</td>
</tr>
<tr>
<td>RS232</td>
<td>10</td>
<td>130</td>
<td>159</td>
<td>184</td>
<td>233</td>
</tr>
<tr>
<td>BasicRSA</td>
<td>4</td>
<td>81</td>
<td>93</td>
<td>119</td>
<td>139</td>
</tr>
</tbody>
</table>

### Cryptoplatform (CPU + memory + 5 crypto cores)

<table>
<thead>
<tr>
<th>Name</th>
<th># Diff. Instances</th>
<th>Min. # Blocks</th>
<th>Max. # Blocks</th>
<th>Min. # Edges</th>
<th>Max. # Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto</td>
<td>6</td>
<td>4402</td>
<td>4424</td>
<td>6503</td>
<td>6537</td>
</tr>
</tbody>
</table>
Experimental Results
Quantitative Evaluation

<table>
<thead>
<tr>
<th>Trust-HUB Benchmarks</th>
<th>Family</th>
<th>[A]</th>
<th>[B]</th>
<th>[C]</th>
<th>[C]*</th>
<th>This work</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AES</strong></td>
<td>3/18</td>
<td>9/18</td>
<td>0/18</td>
<td>18/18</td>
<td>18/18</td>
<td></td>
</tr>
<tr>
<td><strong>RS232</strong></td>
<td>0/10</td>
<td>0/10</td>
<td>9/10</td>
<td>10/10</td>
<td>10/10</td>
<td></td>
</tr>
<tr>
<td><strong>BasicRSA</strong></td>
<td>0/4</td>
<td>2/4</td>
<td>4/4</td>
<td>4/4</td>
<td>4/4</td>
<td></td>
</tr>
</tbody>
</table>


C → [S. K. Haider et al., “HaTCh: Hardware Trojan Catcher”, ‘14]

* Assuming they are activated during the learning phase
## Experimental Results

### Qualitative Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Matches</th>
<th>Conf$_{HT}$</th>
<th>Conf$_{MAX}$</th>
<th>False+</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-T800</td>
<td>9</td>
<td>0.93</td>
<td>0.65</td>
<td>0</td>
<td>5.04</td>
</tr>
<tr>
<td>AES-T1400</td>
<td>81</td>
<td>0.99</td>
<td>0.69</td>
<td>0</td>
<td>4.85</td>
</tr>
<tr>
<td>AES-T1900</td>
<td>11</td>
<td>0.97</td>
<td>0.72</td>
<td>0</td>
<td>4.82</td>
</tr>
<tr>
<td>RS232-T100</td>
<td>7</td>
<td>0.36</td>
<td>0.50</td>
<td>2</td>
<td>4.12</td>
</tr>
<tr>
<td>BasicRSA-T100</td>
<td>4</td>
<td>0.25</td>
<td>0.25</td>
<td>3</td>
<td>1.13</td>
</tr>
</tbody>
</table>

(Full results in the paper or in the poster)
# Experimental Results

## Qualitative Evaluation

<table>
<thead>
<tr>
<th>Name</th>
<th>Matches</th>
<th>Conf$_{HT}$</th>
<th>Conf$_{MAX}$</th>
<th>False+</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto-T000</td>
<td>23</td>
<td>N/A</td>
<td>0.35</td>
<td>N/A</td>
<td>11.80</td>
</tr>
<tr>
<td>Crypto-T100</td>
<td>34</td>
<td>0.81</td>
<td>0.39</td>
<td>0</td>
<td>12.88</td>
</tr>
<tr>
<td>-</td>
<td>34</td>
<td>0.72</td>
<td>0.39</td>
<td>0</td>
<td>12.88</td>
</tr>
<tr>
<td>Crypto-T200</td>
<td>31</td>
<td>0.96</td>
<td>0.71</td>
<td>0</td>
<td>13.43</td>
</tr>
<tr>
<td>Crypto-T300</td>
<td>42</td>
<td>0.88</td>
<td>0.29</td>
<td>0</td>
<td>15.03</td>
</tr>
<tr>
<td>Crypto-T400</td>
<td>34</td>
<td>0.90</td>
<td>0.50</td>
<td>0</td>
<td>15.67</td>
</tr>
</tbody>
</table>
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