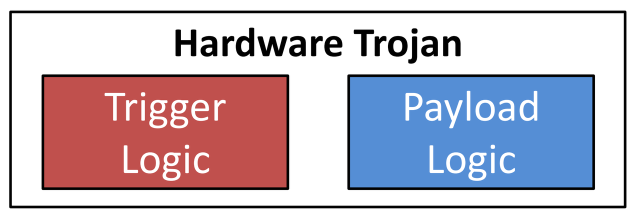
Efficient Control-Flow Subgraph Matching for Detecting Hardware Trojans in RTL Models

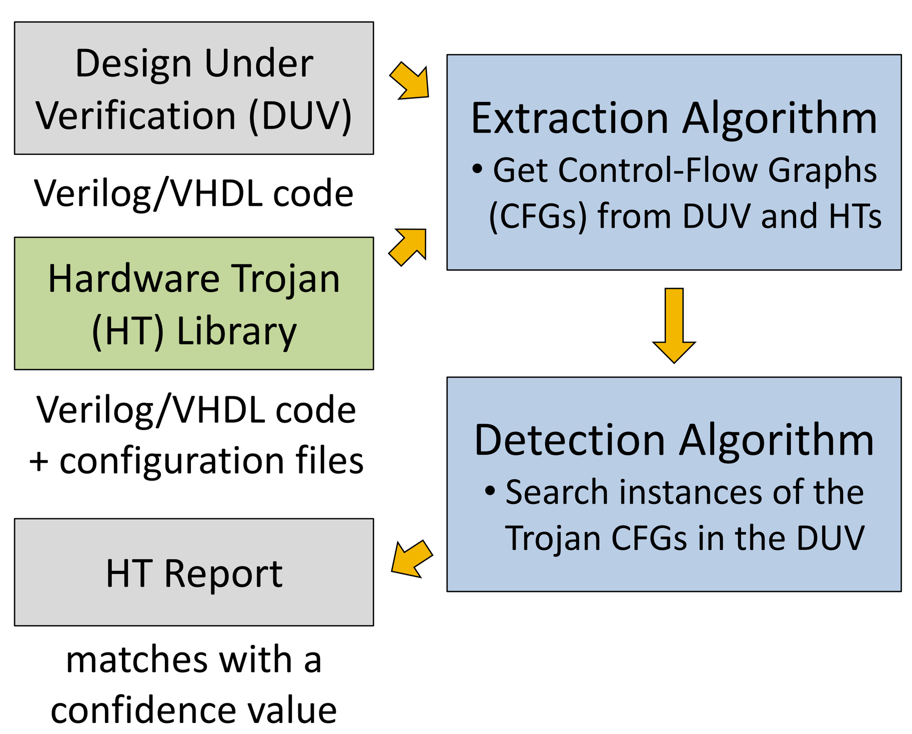
Luca Piccolboni[1,2], Alessandro Menon[2], Graziano Pravadelli[2]
[1] Columbia University, New York, NY, USA
[2] University of Verona, Verona, Italy

Hardware Trojans: An Incoming Threat

- **Hardware Trojans** are defined as malicious and intentional alterations of an integrated circuit that result in undesired behaviors
  - **trigger logic:** activates the malicious behavior under specific conditions
  - **payload logic:** implements the malicious behavior (affects functionality)



Our Approach: Control-Flow Subgraph Matching



Three main contributions:

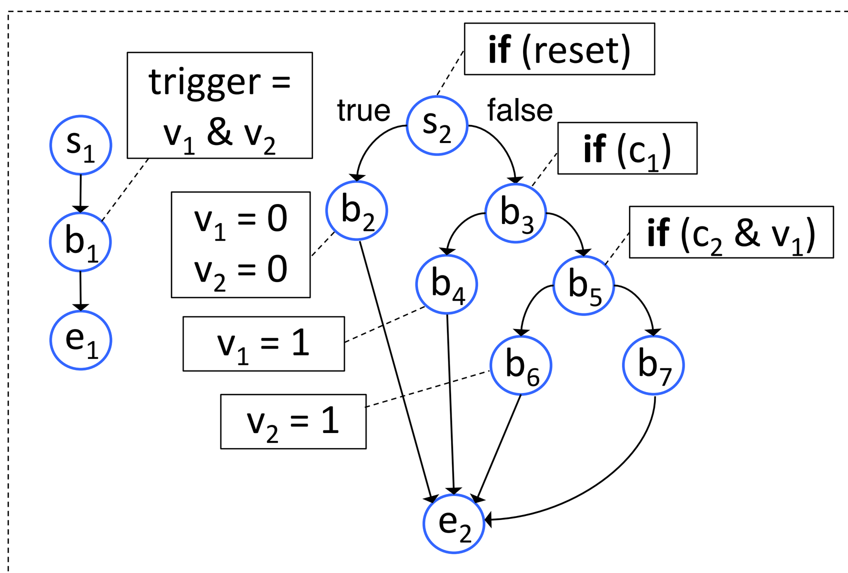
- Hardware Trojan (HT) Library containing parametrizable HTs
- Extraction Algorithm to obtain a CFG from the DUV and the HTs
- Detection Algorithm to identify and locate the HTs in the DUV

In which situations is this useful?

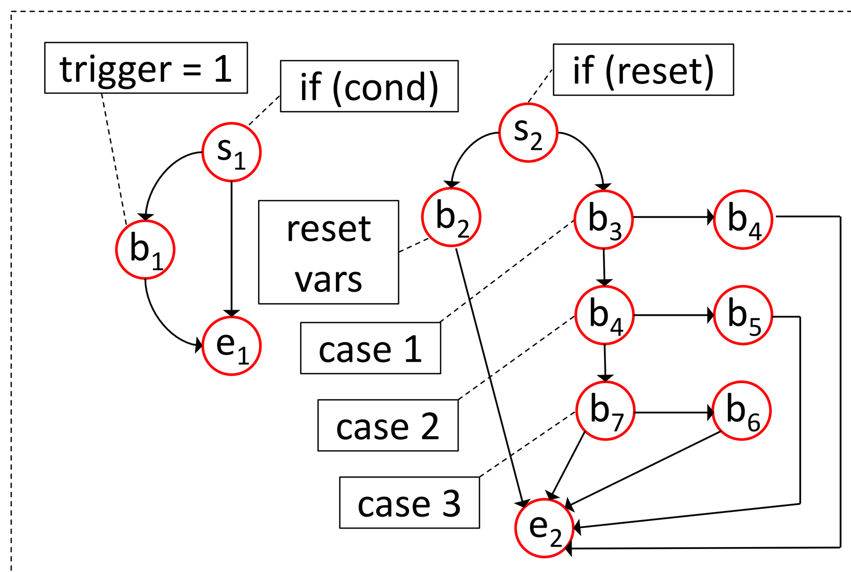
- verify in-house designs at RTL
- verify third-party RTL modules
- verify the results of CAD tools

Hardware Trojan Library

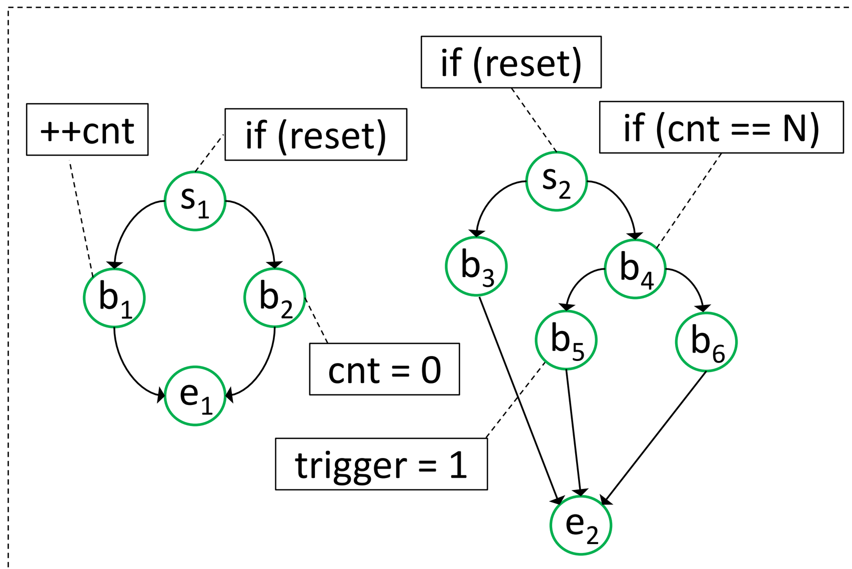
- The library includes the RTL code of known HT **triggers** and **payloads**



Trigger #1: **Cheat Code**



Trigger #2: **Dead Machines**



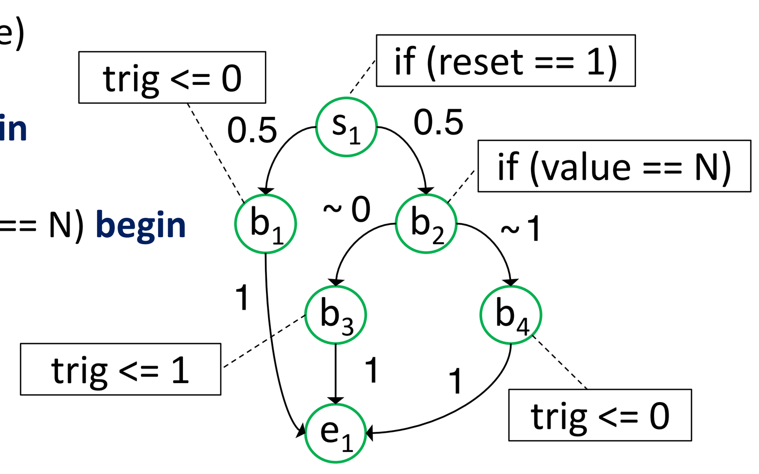
Trigger #3: **Ticking Timebomb**

- Each trigger can be **parametrized** with a configuration file that specifies how to extend the CFG to represent other camouflaged instances of the trigger (by using the **extension directives**)
- The structural characteristics of each trigger are used during the matching (by using the **confidence directives**)
- The payloads can be used as another metric to calculate the confidence

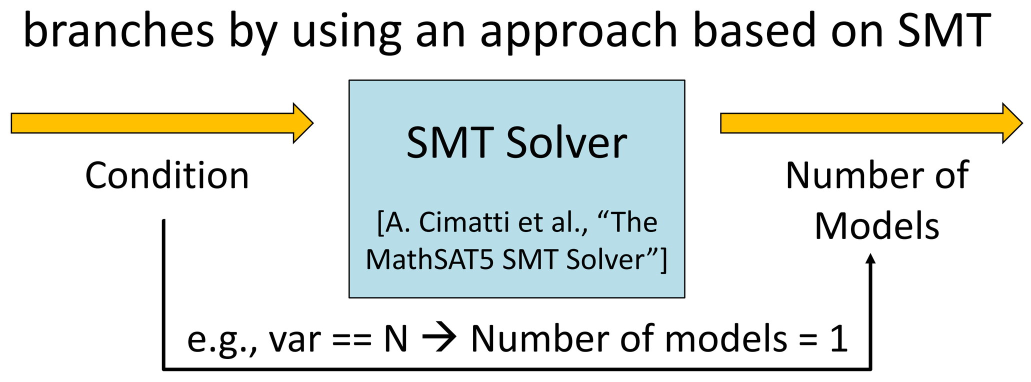
Extraction Algorithm

```
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
```



- The extraction of the CFG is language independent
- The algorithm extracts also the **probabilities** of branches by using an approach based on SMT



- This approach is **scalable** because conditions are usually composed of few variables, and sometimes they can be resolved without using the SMT solver

Detection Algorithm

```
procedure match-trigger(duv, HTLibrary)
   targets = extract-cfg(duv);
   foreach trigger in HTLibrary do
      pattern = extract-cfg(trigger); count = 0;
      while count < counter.getMaxBound() do
         match U= search(pattern, targets);
         pattern.augmentSize(); count++;
   calculate-conf(duv, matches, HTLibrary);
```

- The match is purely based on the **structure** of the CFGs: instructions are **not** considered in any way
- The trigger in the HTLibrary is extended with the extension directives to find camouflaged variants
- Each match is evaluated with the *calculate-conf* procedure to determine if it is a false positive
- **Evaluation of the Algorithm Complexity:**

$$O(l_{HT} * b_{HT} * C(n_{DUV}, n_{HT}))$$

number of triggers in the HTLibrary; max number of extensions; subgraph isomorphism complexity (nodes in DUV and in the HT)

Determining the Confidence

```
procedure calculate-conf(duv, matches, HTLibrary)
   foreach payload in HTLibrary do
      payloads U= extract-cfg(payload);
   foreach match in matches do
      match.conf  = α1 * check-variables(match);
      match.conf += α2 * check-resetlogic(match);
      match.conf += α3 * check-probabilities(match);
      match.conf += α4 * check-payloads(match, duv);
```

- presence of variables: verify if the match uses a variable in the same way of the corresp. pattern
- presence of reset logic: verify if the match has a reset logic similar to that of the corresp. pattern
- average distance of the probabilities: distance of the probabilities of the edges in the match and the probabilities of the edges in the trigger
- degree of dependence between the match and an affine payload: verify if there are variables both in the match and in one of the payloads

$$\beta = c_1\alpha_1 + c_2\alpha_2 + c_3\alpha_3 + c_4\alpha_4$$

- conf: linear combinations of those conditions

Experimental Results

- All the benchmarks are injected with one HT, except Crypto-T000 that has zero HTs and Crypto-T100 that has two HTs.

	Others			Proposed Approach					
Name	[A]	[B]	[C]	Found	Mat.	C_{HT}	C_{MAX}	FP	T(s)
AES-T400	no	no	#	yes	3	0.95	0.64	0	5.04
AES-T500	no	no	#	yes	7	0.93	0.68	0	4.80
AES-T600	no	yes	#	yes	5	0.93	0.41	0	5.12
AES-T700	yes	yes	#	yes	5	0.85	0.50	0	5.11
AES-T800	yes	yes	#	yes	9	0.93	0.65	0	5.04
AES-T900	no	yes	#	yes	7	0.95	0.62	0	4.78
AES-T1000	no	yes	#	yes	4	1.00	0.64	0	4.76
AES-T1100	yes	yes	#	yes	5	0.94	0.47	0	5.67
AES-T1200	no	yes	#	yes	4	0.96	0.54	0	4.69
AES-T1300	no	no	#	yes	82	1.00	0.65	0	5.62
AES-T1400	no	no	#	yes	81	0.99	0.69	0	4.85
AES-T1500	no	no	#	yes	83	0.98	0.65	0	5.80
AES-T1600	no	no	#	yes	7	0.96	0.54	0	4.86
AES-T1700	no	no	#	yes	3	0.98	0.63	0	5.38
AES-T1800	no	no	#	yes	9	1.00	0.69	0	4.86
AES-T1900	no	no	#	yes	11	0.97	0.72	0	4.82

	Others			Proposed Approach					
Name	[A]	[B]	[C]	Found	Mat.	C_{HT}	C_{MAX}	FP	T(s)
AES-T2000	no	yes	#	yes	6	0.93	0.41	0	4.56
AES-T2100	no	yes	#	yes	5	0.95	0.75	0	4.75
RS232-T100	no	no	yes	yes	7	0.36	0.50	2	4.12
RS232-T200	no	no	#	yes	8	0.92	0.56	0	3.13
RS232-T300	no	no	yes	yes	6	0.92	0.31	0	2.74
RS232-T400	no	no	yes	yes	8	0.56	0.51	0	2.32
RS232-T500	no	no	yes	yes	6	0.93	0.31	0	2.80
RS232-T600	no	no	yes	yes	11	0.67	0.35	0	2.39
RS232-T700	no	no	yes	yes	11	0.67	0.53	0	2.58
RS232-T800	no	no	yes	yes	7	0.36	0.50	2	3.23
RS232-T900	no	no	yes	yes	11	0.67	0.52	0	2.43
RS232-T901	no	no	yes	yes	11	0.67	0.52	0	2.48
BasicRSA-T100	no	yes	yes	yes	4	0.25	0.25	3	1.13
BasicRSA-T200	no	no	yes	yes	3	0.25	0.25	1	1.45
BasicRSA-T300	no	yes	yes	yes	4	1.00	0.42	0	1.41
BasicRSA-T400	no	no	yes	yes	5	0.96	0.52	0	1.46

	Proposed Approach					
Name	Found	Mat.	C_{HT}	C_{MAX}	FP	T(s)
Crypto-T000	No	23	N/A	0.35	N/A	11.80
Crypto-T100	yes	34	0.81	0.39	0	12.88
-	yes	34	0.72	0.39	0	12.88
Crypto-T200	yes	31	0.96	0.71	0	13.43
Crypto-T300	yes	42	0.88	0.29	0	15.03
Crypto-T400	yes	34	0.90	0.50	0	15.67

	Benchmarks			
Name	B_{MIN}	B_{MAX}	E_{MIN}	E_{MAX}
AES	2101	2150	3160	3236
RS232	130	159	184	233
BasicRSA	81	93	119	139
Crypto	4402	4424	6503	6537

B_{MIN}/B_{MAX}: min/max number of blocks
E_{MIN}/E_{MAX}: min/max number of edges
AES, RS232, BasicRSA are from TrustHUB, Crypto is from OpenCores

[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", 2014, #: means depends if activated in the learning phase, Mat.: number of matches, C_{HT}: confidence of the HT, C_{MAX}: the highest confidence among false positives, FP: number of false positives, T(s): time in sec.

Take-Home Message

Adopting an approach based on Control-Flow Subgraph Matching is effective and efficient for detecting Hardware Trojans at RTL