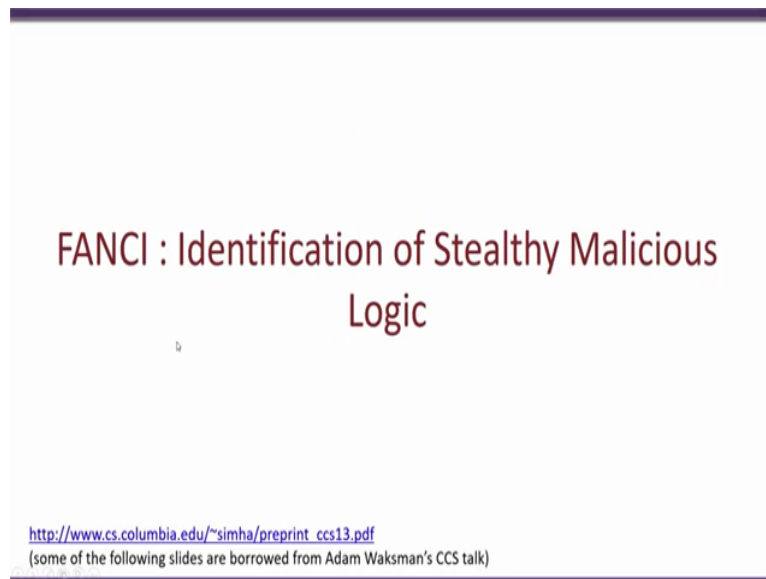


**Information Security - 5 - Secure Systems Engineering**  
**Professor Chester Rebeiro**  
**Indian Institute of Technology, Madras**  
**FANCI: Identification of Stealthy Malicious Logic**

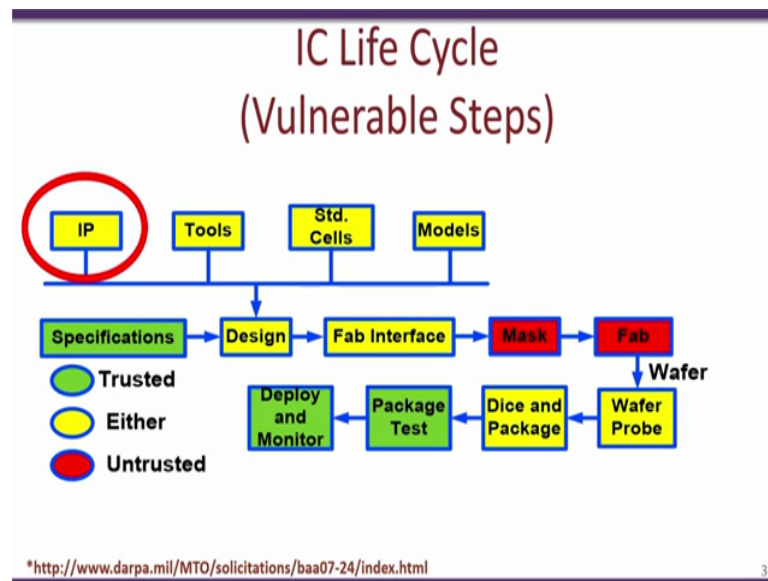
Hello and welcome to this video lecture in the course for secure systems engineering. In this video lecture we will be looking at a particular technique known as fanci, which is a technique used to identify locations within IP used in a hardware design which could potentially have a hardware Trojan present in it.

(Refer Slide Time: 0:39)



So the paper is known as fanci identification of stealthy malicious logic, so some of the slides that are used in this video are borrowed from Adam Waksman's CCS talk, so Adam Waksman is the first author of this paper that was published.

(Refer Slide Time: 1:04)



So in the last video lecture what we had seen was that during the entire design and manufacture cycle during the entire design and manufacture of a hardware IC there can be many phases on many ways through which a hardware Trojan could be inserted in the design. So we in fact had looked at this particular slide wherein we actually discussed that the only trusted components in this entire design flow is at the time of specification and after the device is fabricated and there is a packaged testing that is done and monitoring.

So every other stage during this life's IC design cycle could potentially be spot where a Trojan can be inserted. In this particular talk we will be looking at Trojans that could potentially inserted in third party IP cores. Typically when a company wants to design a hardware IC, they do not actually code every single component of that IC, but rather they would use a system on chip concept and purchase IP cores from several different vendors and then integrate all of these cores within a single chip.

So for instance we have a company which wants to actually design say a communication IC and the communication IC would have several components like a processor core, it may have a DSP core, it may have several IO units like a serial port it, may have various other transceivers, it may have audio input codecs and so on. So what the company would do is that it would purchase IP cores for each of these components and IP core for the general purpose processor another IP core maybe from another company, for the DSP processor if all the various peripherals such as the serial port USB the codec and so on would all be purchased from different manufacturing entities.

Now all of these IP cores would be integrated into the design and then used to manufacture the IC. The reason this is actually done is that it drastically reduces development time because now all that is required by this company is to essentially integrate various IP cores which is purchased. Also this technique of development also reduces the cost drastically because the company would not need to invest in developers to develop each and every component that is used in that IC.

So in the long run this methodology for designing ICs would be extremely beneficial to reduce development time as well as bring down cost. So what we will be looking at today is the threat that each of these IP cores could potentially have a hardware Trojan present in them. So each of these IP cores is designed or purchased from a different third party IP house and each of them could potentially insert a hardware Trojan.

(Refer Slide Time: 4:41)

The slide is titled "Trojans in IPs" in a red font. It contains a bulleted list on the left and a code block on the right. The list points out that third-party IPs are hard to trust and that developers can't search through thousands of lines of code for trojans. The code block shows a Verilog snippet where a register is set to a specific value based on a carry signal, which is a classic hardware Trojan technique.

## Trojans in IPs

- Third party IPs:
  - Can they be trusted?
  - Will they contain malicious backdoors
- Developers don't / can't search 1000s of lines of code looking out for trojans.

```
.  
.  
.  
assign bus_x87_i = arg0 & arg1;  
always @(posedge clk) begin  
  if (rst) data_store_reg7 <= 16'b0;  
  else begin  
    if (argcarry_i37 == 16'hbacd0013) begin  
      data_store_reg7 <= 16'd7777;  
    end  
    else data_store_reg7 <= data_value7;  
  end  
end  
assign bus_x88_i = arg2 ^ arg3;  
assign bus_x89_i = arg4 | arg6 nor arg5;  
.  
.  
.
```

These third party IP core design companies cannot be trusted and therefore they could potentially have malicious Trojans present in them. Now the company which is actually designing this IC (would not) it would not be very easy for them to actually scan through thousands of lines of IP cores looking for potential triggers and potential payloads that may be present.

A complicated IP core would run into like tens of thousands of lines of HDL code and a very small subset of these lines could potentially be having a hardware Trojan. So therefore just by actually looking at the code of a particular IP, it is very difficult to actually detect the presence of a hardware Trojan.

(Refer Slide Time: 5:36)

## FANCI : Identification of Stealthy Malicious Logic

- FANCI: evaluate hardware designs automatically to determine if there is any possible backdoors hidden
- The goal is to point out to testers of possible trojan locations in a huge piece of code

```
.
.
.
assign bus_x87_j = arg0 & arg1;
always @(posedge clk) begin
  if (rst) data_store_reg7 <= 16'b0;
  else begin
    if (argcarry_i37 == 16'hbacd0013) begin
      data_store_reg7 <= 16'd7777;
    end
    else data_store_reg7 <= data_value7;
  end
end
end
assign bus_x88_j = arg2 ^ arg3;
assign bus_x89_j = arg4 | arg6 nor arg5;
.
.
.
b
```

[http://www.cs.columbia.edu/~simha/preprint\\_ccs13.pdf](http://www.cs.columbia.edu/~simha/preprint_ccs13.pdf)

(some of the following slides are borrowed from Adam Waksman's CCS talk)

39

What fanci does is that it provides an automated framework which could highlight regions of this code, it could identify signals present within an IP code which could potentially be areas where a Trojan may be inserted. The whole objective of this framework is to aid the whole objective of this entire framework is to aid the tester to look at particular regions in this code and look more closely at these regions which could potentially have a hardware Trojan in them.

(Refer Slide Time: 6:11)

## Trojans are Stealthy

- **Small**
  - Typically a few lines of code / area
- **Stealth**
  - Cannot be detected by regular testing methodologies (rare triggers)
  - Passive when not triggered

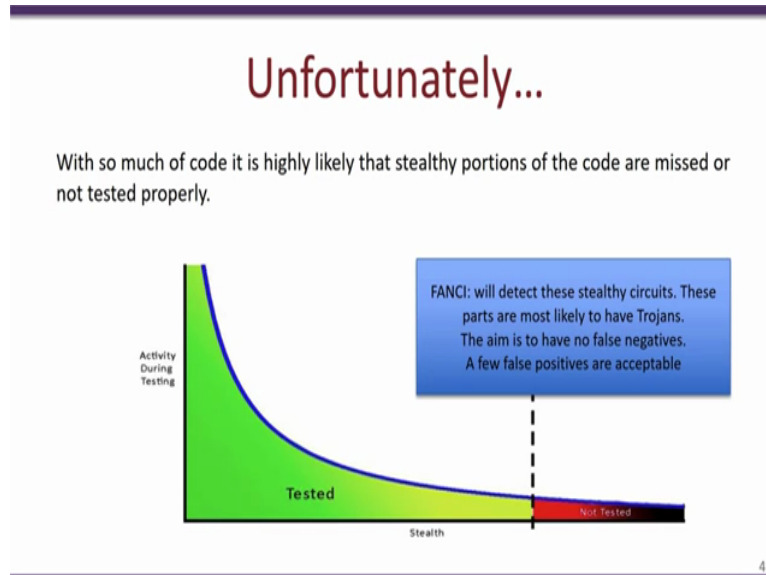
b

40

As we discussed in the previous video, one thing that was quite characteristic of every hardware Trojan is that it is small, typically a few lines of code occupying a very less area in the entire design of the IC and secondly it is stealthy, stealthy means that it is mostly passive

during the normal working of the device or the IC as well as during most of the testing process the Trojan is a passive and inactive and the Trojan only gets activated when specific triggers are obtained.

(Refer Slide Time: 6:53)



Now these two characteristics of being small and stealthy make Trojans very difficult to detect during the regular testing phase of an IC, typically when an IC is tested the testing algorithms are designed in such a way so as to obtain a very high coverage. So we would have a lot of very efficient or testing algorithms which would a boost off a coverage of say 99 percent, what this means is that 99 percent of the chip is actually tested by these algorithms, unfortunately the remaining 1 percent is the areas which are not tested. So it is in this region that a hardware Trojan is likely to be present.

So this particular graph shows how the stealth varies with the activity during testing for areas within the code which are highly active during testing those parts of the code in this IP are not stealthy at all, so they would come into this region of the graph as the stealth increases what we see is that the activity of these components during the testing phase is less. The way fanci actually differs from the other regular testing mechanisms is that fanci focuses on this particular area which normal or regular testing mechanisms would not actually cater to.

Fanci identifies signals in a device which are highly stealthy, so these signals are provided as the output of fanci and it is these signals which should potentially hold a hardware Trojan. The entire aim of fanci is to completely have no false negatives, which means that if a hardware Trojan is present in a device a fanci should be able to detect it. It is okay to have a

few false positives, rather the false positives are acceptable this means that it is okay for fanci to flag a few signals to as having a hardware Trojan when indeed there is no such Trojan present in them.

(Refer Slide Time: 9:14)

### Control Values

By how much does an input influence the output O?

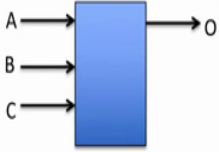
A	B	C	O
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

42

So the main principle by which fanci identifies stealthy signals within a huge IP code is by defining something known as control values. Let us say we have a particular IP core over here, which takes three inputs A, B and C and provides one output of O, so the truth table for this particular hardware design is as shown over here. So we see that the output O varies depending on the inputs A, B and C what fanci actually measures is how much each of these inputs have on the output O, in other words how much does the input A affect the output O and so on.

(Refer Slide Time: 9:50)

### Control Values



By how much does a input influence the output O?

A : has a control of 0 on the output

A	B	C	O
0	0	0	0
1	0	0	0
0	0	1	1
1	0	1	1
0	1	0	0
1	1	0	0
0	1	1	0
1	1	1	0

(A does not matter in this function)  
(A is called unaffacting)

44

So let us take for example the input A and fanci actually gives us a value of 0.5 indicating that A has a control of 0.5 on the output, how does fanci actually compute this is by the following. So what it does is that it keeps the remaining inputs constant for example 00 over here and then changes the value of A so 01 and sees if the output actually changes. So over here you see that when B and C are 0s a change in input A also affects the output O. On the other hand if we look at this particular part of the truth table, where B is 0 and C is 1 the change in A has no effect on the output.

So what we see over here is that for certain values of B and C, A influence is the output, while on the other hand for certain other values of B and C, A has no influence on the output. So what fanci actually measures is the probability with which this input A affects the output. So out of these four for instance A affects two of these, so two out of four and therefore A has a control of 0.5 on the output.

Now for example if we just change the truth table a little bit we change the hardware design a little bit and we actually have this and what we see over here is that independent of the value of A there is no change in the output. For example given that B and C are both 0s, a change in A does not influence the output in any of these cases. Thus, in this case we say that A does not affect the output O or we also say that A is an unaffacting input.

(Refer Slide Time: 11:48)

## Control Values for a Trigger in a Trojan

```
if (addr == 0xdeadbeef) then{
  trigger = 1
}
```

A31 has a control value  $1/2^{16}$

A31	A30	...	A2	A1	A0	trigger
0	0	...	0	0	0	0
0	0	...	0	0	1	0
0	0	...	0	1	0	0
0	0	...	0	1	1	0
:	:	:	:	:	:	
1	1	:	1	1	0	1
:	:	:	:	:	:	
1	1	1	1	1	1	0

Easier to hide a trojan when larger input sets are considered

A low chance of affecting the output  
Lends itself to stealthiness →  
easier to hide a malicious code

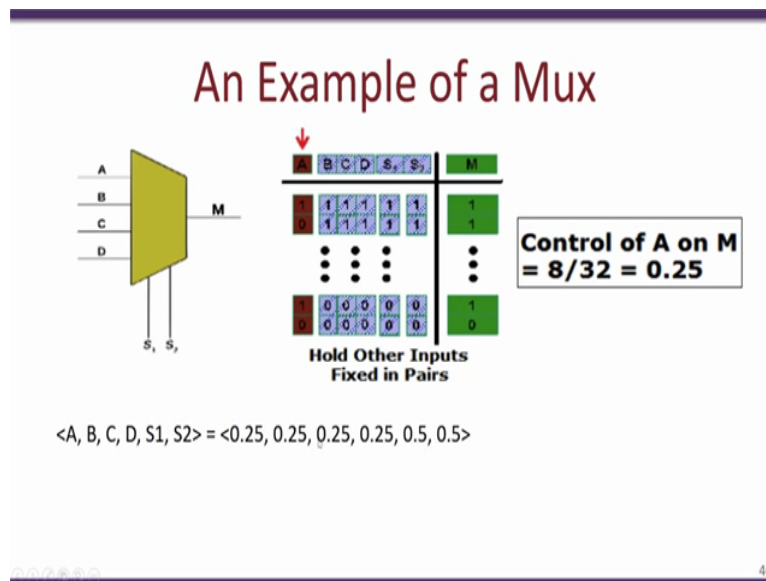
45

So now let us look at how the control values for a trigger in a hardware Trojan looks like. So as we have seen in the previous video a combinational hardware Trojan or a hardware Trojan which is triggered by a cheat code would look something like this, the trigger would wait for a specific input over here or the address should be equal to 0Xdeadbeef and then it would actually set the trigger to be equal to 1. So if we consider the truth table for address and trigger assuming that address is a 32-bit value, it would look something like this, we have the 32 bits of the address A0 to A31 over here and we have the triggered signal which is a single bit over here.

So what we see in this is that in most of the rows in this truth table the trigger has a value of 0 exactly when the values of A0 to A31 has this specific deadbeef input then you have a value of trigger to be 1, in all other cases or trigger has a value of 0. Thus, if we compute the control value for each of these address lines we would get a control value of  $1/2^{16}$ . Now a typical Hardware Trojan would have such a control value that is it would have a such a control value and this is what fancy tries to actually identify given an IP code, such a low control value indicates that these signals are highly stealthy which in turn would indicate that these signals may potentially have a hardware Trojan present in them or in other words these signals may potentially have a trigger for a hardware Trojan present in them.



(Refer Slide Time: 13:45)



Let us take another example of a multiplexer, so this multiplexer is a 4 to 1 multiplexer, it takes 4 inputs A, B, C and D, it has two select lines S 1 and S 2 and it multiplexes one of these inputs to the output M. So depending on the value of S 1 and S 2 either A, B, C or D gets switched into M, as before we can actually write the truth table for this multiplexer and we could also compute the control value for each of these signals.

So there are a total of 4 plus 2 that is 6 input signals and for each of these input signals we can compute the control value. So computing in this way we see that the control of A on M is 8 out of 32 which would mean that 8 times out of 32 A has an influence on the output M, so this use a value of 0.25 as a control of A. Similarly we can actually find the control of all other inputs B, C, D, S 1 and S 2 and these happen to be 0.25, 0.25, 0.25 and 0.5 and 0.5 respectively. Note that S 1 and S 2 have control values of 0.5 each and all other inputs A, B, C and D have a value of 0.25. So in this multiplexer it is highly unlikely that there is a hardware Trojan present in them due to the fact that none of these signals are actually stealthy.

(Refer Slide Time: 15:31)

### An Example of a Malicious Mux

	A	B	C	D	E	S <sub>1</sub>	S <sub>2</sub>	{S <sub>3-66</sub> }
M	0.25	0.25	0.25	0.25	2 <sup>-65</sup>	0.50	0.50	2 <sup>-65</sup>

66 extra select lines which are only modify M when they are set to a particular value

The control values E and S3 to S66 are suspicious because they rarely influence the value of M.  
Perfect for disguising malicious backdoors

Just searching for MIN values is often not enough. Better metrics are needed.

47

Now consider a slight modification to this multiplexer, what we assume is that there is now not just two select lines but there are in fact 65 select lines, the first two are S 1 and S 2 which we have seen as before and besides that we have 63 other select lines which are going to a logic over here which produces either 0 or 1.

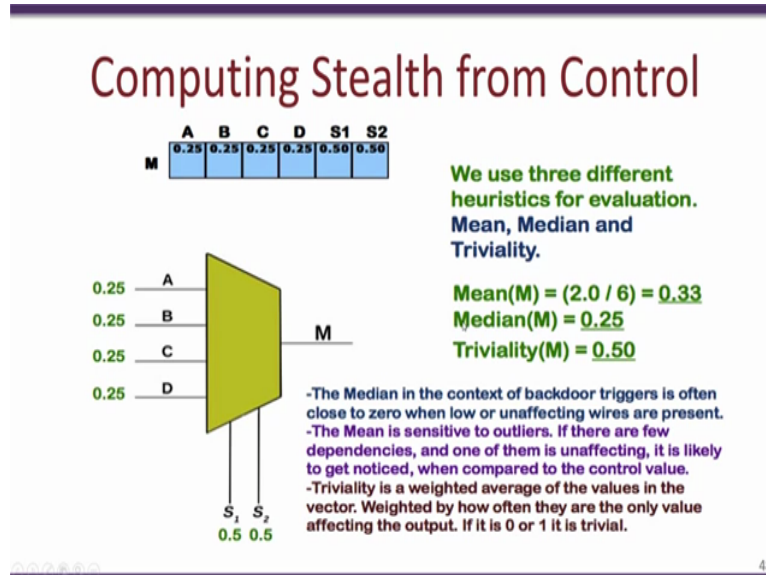
So now let us consider a malicious multiplexer, so what we have done is that we have taken our 4 to 1 multiplexer before and we added a trigger circuit to it. Now this trigger circuit essentially takes 63 inputs and based on the value of this trigger there is a logic which outputs either 0 or 1. Typically, in the passive mode or the output of this logic will be 0 in which case it acts as a regular 4 to 1 multiplexer depending on the values of S 1 and S 2, one of these inputs A, B, C and D would get switched to the output M.

Now depending on this particular 63 bits of additional select lines that gets added and when this output actually gets goes to 1 independent of the values of S 1 and S 2 the input E gets switched into M. So what we see is that this is our hardware Trojan, we have the trigger over here and this E value is what gets leaked to the output of the multiplexer. Now recomputing the control values for these various inputs to the multiplexer we see that A, B, C and D still have a control value of 0.25, the select lines S 1 and S 2 have still have a control value of 0.5.

However, the additional lines which is due to the hardware Trojan has a control value of 2 power minus 63, further the input E has a control value of 2 power minus 65 on the output M. So what this indicates is that the inputs E and the inputs S 3 to S 66 corresponding to these red lines over here are stealthy signals, so they are rarely used and most likely they are

inactive during the general operation of this particular multiplexer and therefore they could be potential venues where a hardware Trojan may be inserted.

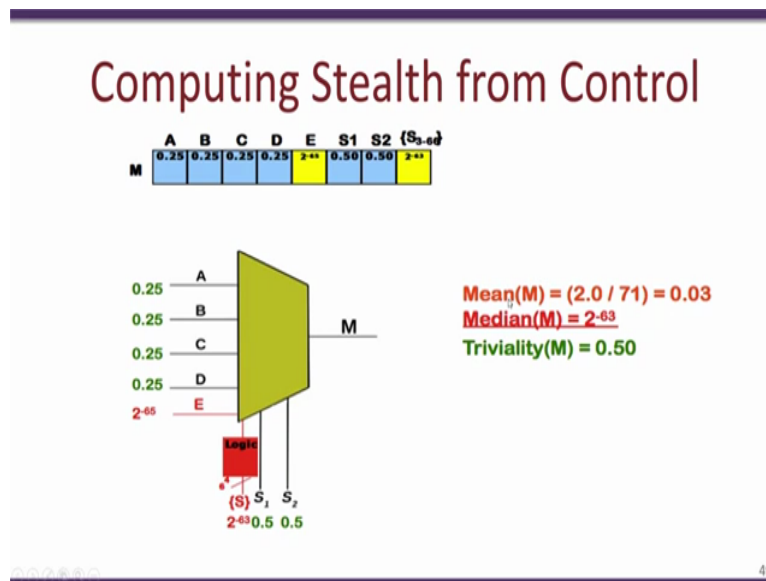
(Refer Slide Time: 18:24)



So what the fancy paper suggests is that they use a set of 3 heuristics a mean, median and something known as triviality to identify potential stealthy signals in a hardware circuit. So each of these three metrics could potentially give you a different result and would become actually active in different scenarios. For example the median in the context of a backdoor triggers is often close to zero when low or unaffacting wires are present, the mean on the other hand is sensitive to outliers if there are few dependencies and one of them is unaffacting it is likely to get noticed, a triviality on the other hand is a weighted average of the values in the vector.

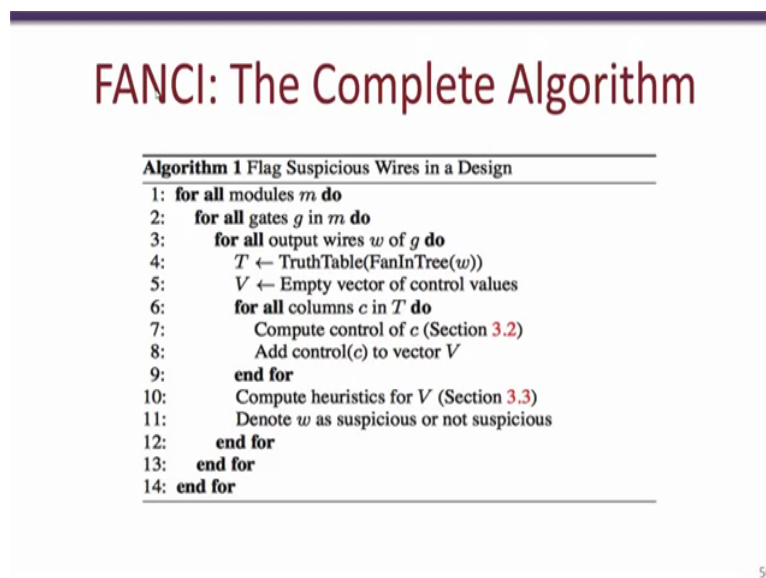
So you could actually go through the paper for more details about how these three metrics differ and how they can be used.

(Refer Slide Time: 19:24)



Now if you actually look at our malicious multiplexer what we see is that the mean has a score of 0.03, the triviality has a score of 0.50 and the median has a score of 2 power minus 63 this median would give you a good indication that this particular circuit possibly has a malicious backdoor present in it.

(Refer Slide Time: 19:47)



So the complete algorithm for fanci is as follows, so as we have discussed initially fanci works at the IP code level, so it is assumed that you have a very large IP core written in RTL like the (( ))(20:00) of VHDL and what fanci does is that it takes this RTL as input and flags all the suspicious wires in the design, so it is these suspicious wires or the stealthy wires which could potentially have a hardware Trojan. What the fanci guarantees is that the

algorithm has no false negatives that is if a Trojan is present in this particular IP then definitely a fanci would actually detect it.

So the algorithm works as follows for each module in the design and then parsing through each gate in that module and for all the wires that are present in the gate, the truth table is built that is T and then control values are computed for each of these inputs, these control values are added to a vector called V and then the three heuristics mean, median and triviality are computed and based on these heuristics, wires are actually set to whether being suspicious or not being suspicious.

So this is a very interesting algorithm and one of the most efficient algorithms to detect potential hardware backdoors present in IP cores and there have been various improvements over this algorithm since 2013, which could detect more advanced backdoors, thank you.