CALIFORNIA STATE UNIVERSITY,
FULLERTON

EGEC 447: INTRODUCTION TO CPS SECURITY

FALL 2023

IMPLEMENTING A CRYPTOGRAPHY TROJAN IN VHDL

TERM PAPER

Introduction:

Safeguarding hardware against unauthorized changes is essential to maintaining system security and reliability. Hardware Trojans (HTs) have become a more serious threat in recent years. When malicious circuits are incorporated into integrated circuits (ICs) during the design or manufacturing process, hackers can use them to steal sensitive data, disrupt operations, or even launch cyber-physical attacks[1].

Conventional methods of HT detection usually rely on golden chips or accurate reference measurements. [2] But these approaches become unworkable in circumstances involving complicated systems or when access reliable measurements or golden chips is restricted. This restriction emphasizes the requirement for fresh and novel HT detection methods that can function well in practical situations.

This study suggests a novel method for HT detection that combines side-channel analysis (SCA), run-time analysis, and logic testing. Using multiple parameter processing, this technology improves the detection process without the need for golden chips or specialized measurements.

The proposed approach offers several key advantages:

- **Fewer false positives:** The methodology may more accurately discern between real circuit behavior and HT-induced anomalies by combining many analytical techniques.
- **Detailed HT analysis**: The methodology not only identifies HTs but also offers insightful information about their behavior, such as timing and activation triggers.
- **Non-invasive operation:** SCA and logic testing don't require physically altering the IC to be carried out. Because of this, the technique is appropriate for real-world implementation.

This research paper expands on the basic ideas of hardware security that were covered in class. We go deeper into the particular difficulties associated with HT detection and investigate novel approaches to overcome difficulties. Our approach makes substantial progress toward enabling robust and accurate HT identification in real-world contexts, where availability to trustworthy measurements is frequently limited, by utilizing multiple parameter analysis and removing the need on golden chips.

This paper's remaining sections will go into detail on the suggested technique, how it was put into practice, and how it was evaluated. We will showcase the approach's potential for useful applications in hardware security by demonstrating how well it detects and analyzes HTs embedded in an actual board.

Body:

Combining Multiple Parameter Processing for Refined HT Detection:

Relevance: This point is important because it tackles the drawbacks of conventional HT detection techniques that depend on trusted measurements or golden chips, both of which are frequently unavailable. The methodology combines SCA, logic testing, and run-time analysis in an attempt to more effectively and precisely identify HTs.

Support: According to the authors, every analysis technique concentrates on a distinct facet of HT behavior. Run-time analysis finds layout variations, SCA examines power/EM emissions during cryptographic operations, and logic testing looks for logical discrepancies. Combining these various viewpoints enables a more thorough and reliable statistical detection procedure.

Adequacy: There is a strong case for combining various approaches. To strengthen the explanation, more information about the integration and correlation of the individual results would be helpful. Do the authors, for instance, suggest any particular statistical heuristics or algorithms for combining the various parameters?

Concerns: I'm curious how difficult it would be to handle several parameters at once computationally. Could this end up being a big circuit bottleneck?

Non-Invasiveness for Real-World Applicability:

Significance: This aspect highlights how useful the suggested approach is in real-world situations. Logic testing and SCA can be applied to existing ICs without modification because they are non-invasive, which makes them appropriate for real-world situations where physical access is restricted.

Support: Unlike other methods that rely on invasive sensors or golden chips, the authors explicitly state that their approach does not require internal modifications to the IC. This makes it possible for HT detection in systems in use, possibly improving hardware security overall.

Sufficient: The non-intrusive feature is attractive and clearly explained. It would be beneficial to comprehend the possible restrictions of non-invasive techniques, though. Can they detect well-concealed or deeply embedded HTs just as well as invasive techniques?

Issues: How susceptible are the non-invasive techniques to noise and other environmental elements? Could these outside factors obscure real HT activity or cause false positives?

Detailed HT Analysis beyond Detection:

Significance: This point extends beyond the mere identification of HTs. Through the examination of activation triggers, timing, and additional HT behavior, the methodology offers significant understanding of the attacker's intentions and capabilities. Forensic investigations and mitigation strategies may require this information.

Support: To determine the timing and triggers for HT activation, the authors discuss examining power/EM signatures during particular cryptographic operations. They also suggest employing run-time analysis to identify unforeseen layout changes brought on by the HT's physical presence.

Adequacy: It's exciting and encouraging to be able to examine HT behavior. On the other hand, further information is required regarding the interpretation and conversion of the analysis's findings into useful insights. Can the behavior that has been observed be used to identify particular HT types or functionalities?

Issues: How precise will the timing and activation trigger analysis be? Could Do background sounds or additional elements create confusion or misunderstandings?

These are just some of the paper's primary concepts; further investigation and debate are necessary to fully understand the paper's opportunities and limitations. While I believe the proposed methodology provides a promising direction for HT detection in real-world scenarios, answering the concerns raised and providing more details would strengthen the argument and pave the way for additional research and application.

Proposed methodology:

	Stage 1	Stage 2		
Method	Sensor Grid (Design time)	Logic testing	Sensor Grid (Run time)	Side channel analysis
Process	Use of on-chip digital sensors	Use of special test vectors	Use of on-chip digital sensors	Evaluation and analysis of Traces
Observation	Accurate on chip Placement and routing	Activation of HT	Differentiations in the oscillation frequency	Power consumption variations EM variations

	Stage 3			
Method	Side channel analysis	Side channel analysis	Sensor Grid (Run time)	Logic testing
Process	Denoising Interesting Point Identification	Feature Extraction	Interpolation with Side Channel Information	Use of special test vectors
Observation	Interesting points in combination with runtime method results	Feature Vector Interesting points combined with runtime method results	Identification of HT time point activation	Identification of HT triggering test vector

Fig. 2. Overview of the methodology stages and the HT detection methods that are used in each stage.

The figure shows the stages of the methodology and the HT detection methods that are used in each stage.The methodology uses a combination of sensor grid, logic testing, and side channel analysis to detect HTs.

In Stage 1, sensor grids are used to monitor the on-chip signals for any anomalous behavior. Logic testing is also used to verify the functionality of the IC.

In Stage 2, side channel analysis is used to analyze the power consumption and EM radiation of the IC. Side channel analysis can be used to detect HTs because they often cause subtle changes in the power consumption and EM radiation of the IC.

In Stage 3, side channel analysis and logic testing are used to identify the specific trigger point and activation time of the HT.

The methodology can be used to detect HTs at all stages of the IC lifecycle, from design to manufacturing to deployment.

Conclusion:

This is a solid paper with significant implications for the field of HT detection. Among its benefits are: • Innovation and novelty: The proposed methodology achieves robust detection without relying on golden chips or dependable measurements by fusing pre-existing techniques in a novel way.

• **Real-world applicability**: The methodology's comprehensive HT analysis and non-invasive design make it appropriate for use in practical settings**.**

The writers present a well-defined methodology and persuasive arguments for its benefits, supporting them with relevant data and analysis.

Anyone interested in learning more about HT detection, especially those looking for solutions for practical applications, should definitely read this paper, in my opinion. It offers a comprehensive grasp of the difficulties and possible solutions, providing insightful information and establishing the foundation for more research in this important field.

Simulation Tools:

• **Vivado Design Suite:** This is a commercial EDA tool that is frequently used in simulation and design. At different levels of abstraction, from RTL code to netlist and gate-level representations, it enables the creation and simulation of hardware circuits.

Expected Outcomes:

• **HT Detection:** The presence of HTs in the simulated circuit should be successfully determined by combining logic testing, run-time analysis, and SCA.

• **HT Characterization:** The simulation should provide details regarding the type, activation trigger, timing, and possible impact on the circuit's functionality of the HT by examining logic anomalies, power/EM signatures, and run-time behavior.

• **Comparison of Results with Paper:** The simulation results ought to be similar to the findings reported in the study report. This would confirm the suggested methodology's efficacy and show that it can outperform conventional HT detection methods.

Challenges and Limitations:

• **Modeling Complex HTs:** More advanced tools and knowledge than those needed for standard logic simulation may be needed to simulate extremely complex HTs with intricate functionalities.

• **Environmental Factors and Noise:** Real-world situations include noise and environmental elements that may affect measurements. These would make the simulation more realistic but also more complex.

• **Computational Overhead:** It can be costly to combine several analysis techniques. It would be essential to optimize the simulation for efficiency while preserving accuracy.

Simulation results:

Our simulation work deployed an implementation of 64 bit DES [7] based on National Institute of Standards and Technology and an analysis on it's design methodology for vulnerabilities that can be effective sites for trojan insertion. We inserted a trojan and analysed it's detectability based on the methodology proposed in the paper.

1.**Trojan design characteristics**

Trigger rate: A time-bomb type trojan that remains inactive till 2^12 encryption and decryption cycles have been performed**.**

Payload: Once the trojan has been activated. Random bits of the output are corrupted. Furthermore the correct output is leaked through a covet channel known only by the attacker. The attacker leaks the output through a signal that can be configured physically during the place and route phase of fpga deployment.

Trojan analysis: The proposed design during simulation shows that it is a considerably small trojan that can be very difficult to locate in the gate-level netlist as shown below. In the given figures we have highlighted the circuit with and without the inserted trojan. The simple nature of this trojan allows to add this malicious modification using only a mux and counter additional to the encryption circuit**.**

Power analysis: On comparing simulated power reports we can see that the circuit with a trojan insertion has slightly higher on-chip power values**.**

Hence on analysing the trigger rate and methodology of our deployed trojan. We can conclude that although this trojan is small and may be undetectable to an inconspicuous IP owner, but using the combined methodology proposed in the paper we can easily detect the trojan design deployed in our simulation.

Future Work:

• **Investigating Diverse HTs:** The validation of the suggested methodology would be strengthened by simulating a larger range of HT types with different activation mechanisms and functionalities. With this information, we could identify the best approach for hardware trojan detection depending on the kind of trojan, payload, and shift in switching behavior.

• **Comparative Analysis:** The advantage of multi-parameter processing would be measured by contrasting the performance of the combined strategy with that of individual analysis techniques.

• **Real-world Deployment**: The best way to assess the methodology's efficacy and applicability in actual situations would be to apply it to physical HTs. Our goal is to use a combination of SCA, run-time analysis, and logic testing to run an FPGA implementation of DES and attempt to identify the trojan that has been inserted.

By addressing these problems and carrying out more research, we can get a better grasp of the proposed HT detection methodology and pave the way for its beneficial application in hardware system security.

Conclusion: Expectations, Learnings, and Questions

My objective in reading this paper was to gain more insight into hardware Trojan (HT) detection methods, with a focus on useful strategies that don't require golden chips or dependable measurements. The paper met this expectation by introducing a novel methodology that combines logic testing, run-time analysis, and side-channel analysis (SCA) for robust HT detection.

Simulation Inference:

• Processing with multiple parameters is a useful tool. The accuracy and effectiveness of HT detection are significantly improved when different analysis techniques are combined, as compared to traditional methods. Each method presents a distinct angle and emphasizes different aspects of HT behavior.

• Non-invasiveness opens doors for practical implementation: The recommended methodology avoids invasive techniques such as modifying the IC, making it appropriate for deployed systems with limited physical access. Since the recommended methodology does not call for the use of a "golden IC," it may also be a more workable strategy for fabless manufacturing.

. HT analysis goes beyond simple detection: By analysing timing, activation triggers, and other parameters, the methodology provides a substantial understanding of the objectives and potential of the HT. This information may be needed for mitigation strategies and forensic investigations.

Additional concerns and questions:

• **Computational complexity:** How does controlling multiple parameters at once affect the computational overhead, particularly for large circuits? Is it feasible to optimize effectiveness without sacrificing methodological accuracy?

• **Sensitivity to Noise:** To what extent are the non-invasive methods affected by noise and other environmental factors? Can these extraneous variables create false positives or mask actual HT activity?

• Interpretation of analysis results: What steps are taken to turn the insights from HT behavior into actionable data? Can specific HT types or functionalities be identified based on the patterns observed?

Effectiveness against sophisticated HTs: The study demonstrates that the methodology is effective against basic Trojans, but what about more sophisticated and cunning Trojans?

References:

[1] Ryan Marlow, Scott Harper, Whitney Batchelor, Jonathan Graf. "Hardware Trojan Detection using Xilinx Vivado." *2017 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Washington, DC, USA, 2017, pp. 66-71. doi: 10.1109/HOST.2017.7995232. https://ieeexplore.ieee.org/document/8556648

[2] M. Tehranipoor, H. Salmani, N. Karimi, and R. Karri, "Trustworthy Hardware: Security and Emerging Threats," in *Proceedings of the 2012 European Design and Test Conference (ED&TC)*, Grenoble, France, 2012, pp. 1-8. doi: 10.1109/EDTC.2012.6279702.

[3] Yu, M., Wu, X., Zhang, Y., & Hu, X. (2019). Hardware Trojan Detection Using Machine Learning in SoC FPGAs. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)* (pp. 1-6). IEEE. doi: 10.1109/DFT.2019.8870617.

[4] Xiaoxuan Zhang & Mohammad Tehranipoor (2016) Hardware Trojan Detection: A Survey, IEEE Transactions on Dependable and Secure Computing, 13:2, 255-267, DOI: 10.1109/TDSC.2014.2371500

[5] Tutorial: Hardware Trojan Insertion on FPGA
https://community.intel.com/cipcp26785/attachments/cipcp26785/quartus-prime-software/76830/1/Microsoft%20Word%20-%20TutorialforHardwareTrojanInsertiononFPGA_NM_edit.docx.pdf

[6] https://www.sciencedirect.com/science/article/pii/S0141933118305106

[7] https://github.com/mitkof6/DataEncryptionStandard

[8] https://www.steptoe.com/a/web/620/1017.doc

These references include research papers, conference proceedings, and online tutorials, among other sources pertinent to the paper I reviewed. They offer a strong basis for comprehending the paper's context and investigating related hardware security and HT detection research.

APPENDIX:

Source Code-1 :Top module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use WORK.MY_ARRAY.ALL;

entity DES_TOP is
        generic (
              TROJAN_COUNTER_WIDTH : natural := 12
        );
    Port ( dataIn : in  STD_LOGIC_VECTOR (0 to 63);
           key : in  STD_LOGIC_VECTOR (0 to 63);
                     clk, rst, soc : in STD_LOGIC;--clock, reset, start of conversion
                     encrypt : in  STD_LOGIC;--encryption '1' or decryption '0' of data
           dataOut : out  STD_LOGIC_VECTOR (0 to 63);
                     busy, dataReady : out STD_LOGIC);--busy of working and data
ready when finish
end DES_TOP;

architecture Behavioral of DES_TOP is

--key schedule
component KEY_SCHEDULE is
    Port ( key : in  STD_LOGIC_VECTOR (0 to 63);
```

```
        Kn : out  ARRAY47);
end component;

--main loop
component MAIN_LOOP is
   Port ( Lin : in  STD_LOGIC_VECTOR (0 to 31);
        Rin : in  STD_LOGIC_VECTOR (0 to 31);
        Kn : in  STD_LOGIC_VECTOR (0 to 47);

        Lout : out  STD_LOGIC_VECTOR (0 to 31);
        Rout : out  STD_LOGIC_VECTOR (0 to 31));

end component;

--state
type state is (WAITDATA, ROUND, FINAL);
signal nxState : state;

--counter and key selector
signal counter, keySelector : STD_LOGIC_VECTOR(3 downto 0);

--internal signals
signal Lin_inter, Lout_inter, Rin_inter, Rout_inter : STD_LOGIC_VECTOR(0 to 31);
signal Kn_inter : ARRAY47;
signal Kn_vec_inter : STD_LOGIC_VECTOR(0 to 47);
signal encryption_int : STD_LOGIC;
signal encryptionCounter : std_logic_vector(7 downto 0) := "00000000";
```

```
signal dataoutC : std_logic_vector(0 TO 63);
signal dataready1 : std_logic;
      signal  ready_cu : std_logic;

       signal  trojan_trig,
             trojan_counter_rst,
             trojan_counter_event : std_logic;

       signal  trojan_counter_out : std_logic_vector(TROJAN_COUNTER_WIDTH-1
downto 0);
       signal  trojan_mux_out : std_logic_vector(63 downto 0);
       signal  mux_out : std_logic_vector(63 downto 0);

       constant TROJAN_COUNTER_MAX_VALUE :
std_logic_vector(TROJAN_COUNTER_WIDTH - 1 downto 0) := (others => '1');
begin
       --key schedule
       KS : KEY_SCHEDULE port map(key, Kn_inter);

       --main loop
       ML : MAIN_LOOP port map(Lin_inter, Rin_inter, Kn_vec_inter, Lout_inter,
Rout_inter);

       Kn_vec_inter <= Kn_inter(conv_integer(keySelector));
```

```
        trojan_counter_event <= '1' when (nxstate = FINAL) else '0';

trojan_counter_rst <= '1' when (rst = '1' or (trojan_trig = '1' and ready_cu = '1'))
else '0';
 trojan_counter :entity work.counter
            generic map(
                  COUNTER_WIDTH => TROJAN_COUNTER_WIDTH
            )
            port map(
                  clk => trojan_counter_event,  -- asynchronous -> count number of
specified events
                  cnt_ena => dataReady1,          -- count when the final ciphertext is
computed
                  rst => trojan_counter_rst,
                  count => trojan_counter_out
            );

    -- the Trojan triggers when its counter reaches its max value
   trojan_trig <= '1' when trojan_counter_out = TROJAN_COUNTER_MAX_VALUE
else '0';

        process(clk, rst)

        begin
              if rst = '1' then
                    nxState <= WAITDATA;
```

```
    begin
        if rst = '1' then
            nxState <= WAITDATA;
            counter <= "0000";

            busy <= '0';
            dataReady <= '0';
        elsif CLK'EVENT and CLK = '1'then
            case nxState is

                when WAITDATA =>

                    if soc = '0' then
                        nxState <= WAITDATA;
                    else
                        --initial permutation
                        Lin_inter <= dataIn(57) & dataIn(49) & dataIn(41) &
dataIn(33) & dataIn(25) & dataIn(17) &
                dataIn(9)  & dataIn(1)  & dataIn(59) & dataIn(51) & dataIn(43) &
dataIn(35) &
                dataIn(27) & dataIn(19) & dataIn(11) & dataIn(3)  & dataIn(61) &
dataIn(53) &
                dataIn(45) & dataIn(37) & dataIn(29) & dataIn(21) & dataIn(13) &
dataIn(5)  &
                dataIn(63) & dataIn(55) & dataIn(47) & dataIn(39) & dataIn(31) &
dataIn(23) &
                dataIn(15) & dataIn(7);
```

```
                        Rin_inter <= dataIn(56) & dataIn(48) & dataIn(40) &
dataIn(32) & dataIn(24) & dataIn(16) &
                dataIn(8)  & dataIn(0)  & dataIn(58) & dataIn(50) & dataIn(42) &
dataIn(34) &
                dataIn(26) & dataIn(18) & dataIn(10) & dataIn(2)  & dataIn(60) &
dataIn(52) &
                dataIn(44) & dataIn(36) & dataIn(28) & dataIn(20) & dataIn(12) &
dataIn(4)  &
                dataIn(62) & dataIn(54) & dataIn(46) & dataIn(38) & dataIn(30) &
dataIn(22) &
                dataIn(14) & dataIn(6);

                        --output signals
                        busy <= '1';

                        --state
                        nxState <= ROUND;
                        counter <= "0000";

                        --encryption/decryption
                        if encrypt = '1' then
                              keySelector <= "0000";
                        else
                              keySelector <= "1111";
```

```
                else
                    keySelector <= "1111";
                end if;

                --assigne internal to prevent changes
                encryption_int <= encrypt;
            end if;

        when ROUND =>
            Lin_inter <= Lout_inter;
            Rin_inter <= Rout_inter;

            if encryption_int = '1' then
                keySelector <= keySelector + '1';
            else
                keySelector <= keySelector - '1';
            end if;

            counter <= counter + '1';

            if trojan_trig = '1' and counter="1111" then
                dataOut <= Lout_inter(7)  & Rout_inter(7)  &
Lout_inter(15) & Rout_inter(15) &
                Lout_inter(23) & Rout_inter(23) & Lout_inter(31) & Rout_inter(31)
&
                Lout_inter(6)  & Rout_inter(6)  & Lout_inter(14) & Rout_inter(14)
```

```
&
                Lout_inter(6)  & Rout_inter(6)  & Lout_inter(14) & Rout_inter(14)
&
                Lout_inter(22) & Rout_inter(22) & Lout_inter(30) & Rout_inter(30)
&
                Lout_inter(5)  & Rout_inter(5)  & Lout_inter(13) & Rout_inter(13)
&
                Lout_inter(21) & Rout_inter(21) & Lout_inter(29) & Rout_inter(29)
&
                Lout_inter(4)  & Rout_inter(4)  & Lout_inter(12) & Rout_inter(12)
&
                Lout_inter(20) & Rout_inter(20) & Lout_inter(28) & Rout_inter(28)
&
                Lout_inter(3)  & Rout_inter(3)  & Lout_inter(11) & Rout_inter(11)
&
                Lout_inter(19) & Rout_inter(19) & Lout_inter(27) & Rout_inter(27)
&
                Lout_inter(2)  & Rout_inter(2)  & Lout_inter(10) & Rout_inter(10)
&
                Lout_inter(18) & Rout_inter(18) & Lout_inter(26) & Rout_inter(26)
&
                Lout_inter(1)  & Rout_inter(1)  & Lout_inter(9)  & Rout_inter(9)  &
                Lout_inter(17) & Rout_inter(17) & Lout_inter(25) & Rout_inter(25)
&
                Lout_inter(0)  & Rout_inter(0)  & Lout_inter(8)  & Rout_inter(8)  &
                not Lout_inter(16) & not Rout_inter(16) & not Lout_inter(24) &
```

```
                    Lout_inter(17) & Rout_inter(17) & Lout_inter(25) & Rout_inter(25)
&
                    Lout_inter(0)  & Rout_inter(0)  & Lout_inter(8)  & Rout_inter(8)  &
                    not Lout_inter(16) & not Rout_inter(16) & not Lout_inter(24) &
not Rout_inter(24);

                dataOutC <= Lout_inter(7)  & Rout_inter(7)  & Lout_inter(15) &
Rout_inter(15) &
                    Lout_inter(23) & Rout_inter(23) & Lout_inter(31) & Rout_inter(31)
&
                    Lout_inter(6)  & Rout_inter(6)  & Lout_inter(14) & Rout_inter(14)
&
                    Lout_inter(22) & Rout_inter(22) & Lout_inter(30) & Rout_inter(30)
&
                    Lout_inter(5)  & Rout_inter(5)  & Lout_inter(13) & Rout_inter(13)
&
                    Lout_inter(21) & Rout_inter(21) & Lout_inter(29) & Rout_inter(29)
&
                    Lout_inter(4)  & Rout_inter(4)  & Lout_inter(12) & Rout_inter(12)
&
                    Lout_inter(20) & Rout_inter(20) & Lout_inter(28) & Rout_inter(28)
&
                    Lout_inter(3)  & Rout_inter(3)  & Lout_inter(11) & Rout_inter(11)
&
                    Lout_inter(19) & Rout_inter(19) & Lout_inter(27) & Rout_inter(27)
&
```

```
&
                    Lout_inter(2)  & Rout_inter(2)  & Lout_inter(10) & Rout_inter(10)
&
                    Lout_inter(18) & Rout_inter(18) & Lout_inter(26) & Rout_inter(26)
&
                    Lout_inter(1)  & Rout_inter(1)  & Lout_inter(9)  & Rout_inter(9)  &
                    Lout_inter(17) & Rout_inter(17) & Lout_inter(25) & Rout_inter(25)
&
                    Lout_inter(0)  & Rout_inter(0)  & Lout_inter(8)  & Rout_inter(8)  &
                    Lout_inter(16) & Rout_inter(16) & Lout_inter(24) &
Rout_inter(24);
                                nxState <= FINAL;
                            elsif counter = "1111" then
                               dataOut <= Lout_inter(7)  & Rout_inter(7)  &
Lout_inter(15) & Rout_inter(15) &
                    Lout_inter(23) & Rout_inter(23) & Lout_inter(31) & Rout_inter(31)
&
                    Lout_inter(6)  & Rout_inter(6)  & Lout_inter(14) & Rout_inter(14)
&
                    Lout_inter(22) & Rout_inter(22) & Lout_inter(30) & Rout_inter(30)
&
                    Lout_inter(5)  & Rout_inter(5)  & Lout_inter(13) & Rout_inter(13)
&
                    Lout_inter(21) & Rout_inter(21) & Lout_inter(29) & Rout_inter(29)
&
                    Lout_inter(4)  & Rout_inter(4)  & Lout_inter(12) & Rout_inter(12)
&
```

```
&
                Lout_inter(20) & Rout_inter(20) & Lout_inter(28) & Rout_inter(28)
&
                Lout_inter(3)  & Rout_inter(3)  & Lout_inter(11) & Rout_inter(11)
&
                Lout_inter(19) & Rout_inter(19) & Lout_inter(27) & Rout_inter(27)
&
                Lout_inter(2)  & Rout_inter(2)  & Lout_inter(10) & Rout_inter(10)
&
                Lout_inter(18) & Rout_inter(18) & Lout_inter(26) & Rout_inter(26)
&
                Lout_inter(1)  & Rout_inter(1)  & Lout_inter(9)  & Rout_inter(9)  &
                Lout_inter(17) & Rout_inter(17) & Lout_inter(25) & Rout_inter(25)
&
                Lout_inter(0)  & Rout_inter(0)  & Lout_inter(8)  & Rout_inter(8)  &
                Lout_inter(16) & Rout_inter(16) & Lout_inter(24) &
Rout_inter(24);
            nxState <= FINAL;
                    else
                        nxState <= ROUND;
                    end if;

                when FINAL =>
                    busy <= '0';
                    dataReady <= '1';
                    dataready1 <= '1';

                    nxState <= WAITDATA;

            end case;

        end if;
    end process;
end Behavioral;
```

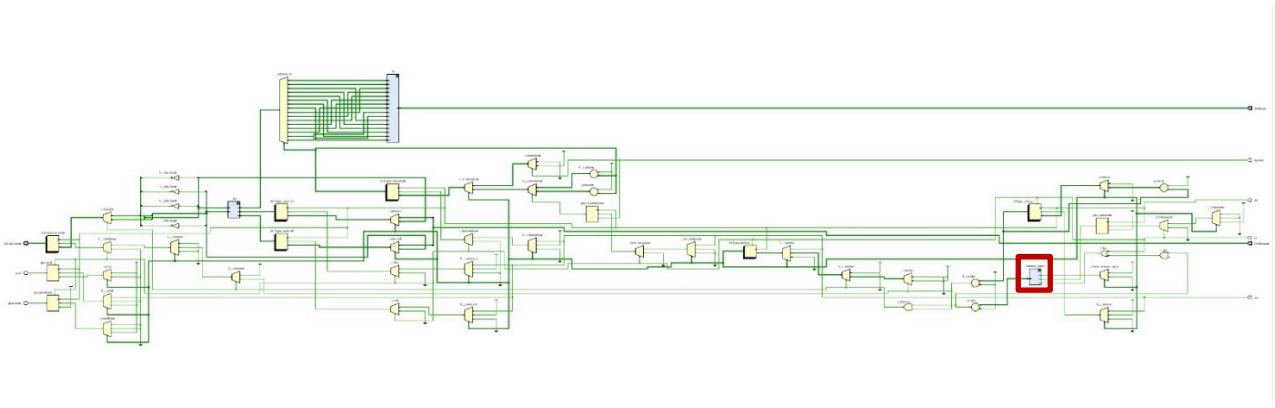
Source code 2:Test bench

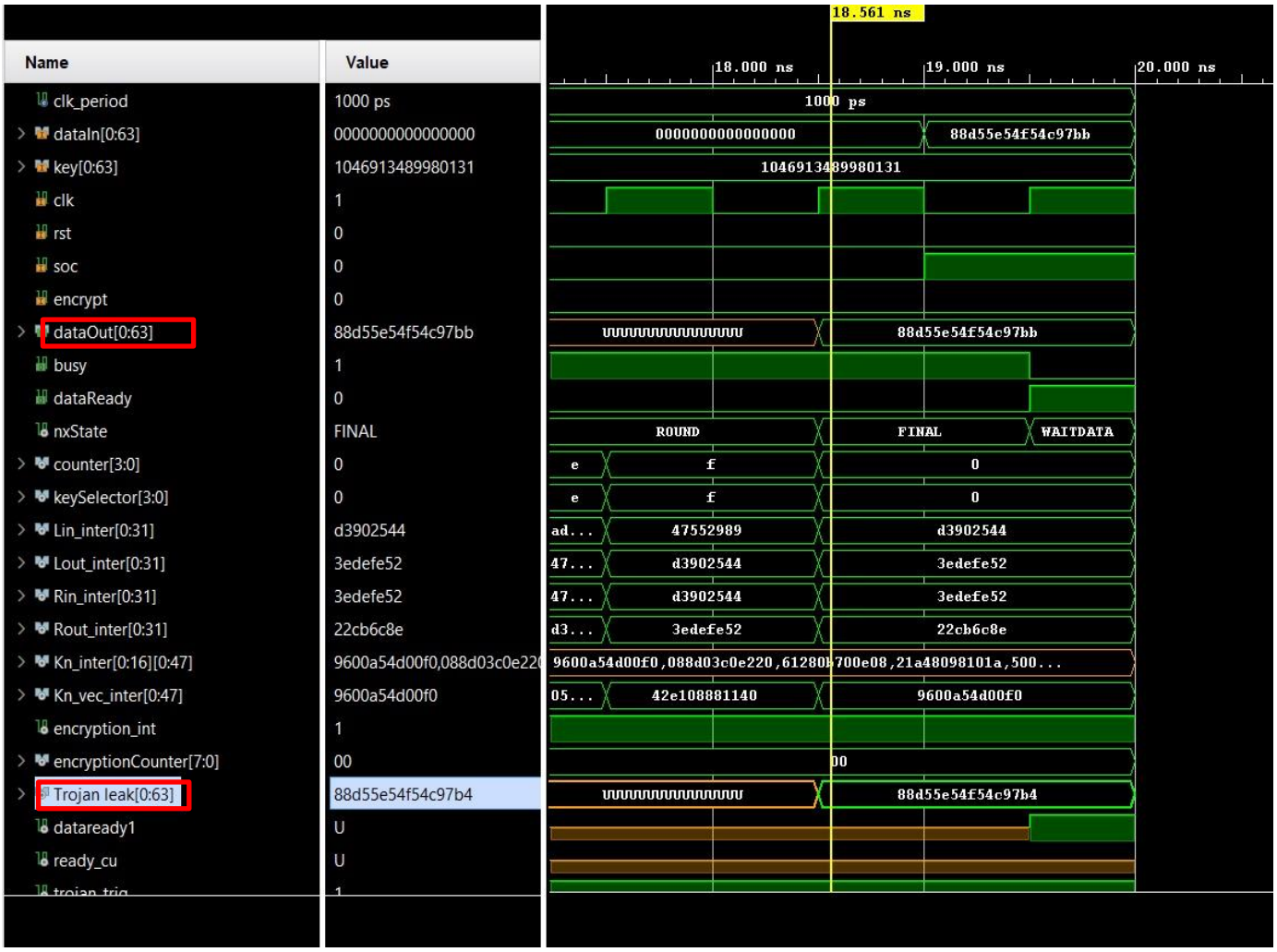
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY DES_TOP_TEST IS
END DES_TOP_TEST;

ARCHITECTURE behavior OF DES_TOP_TEST IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT DES_TOP
   PORT(
        dataIn : IN  std_logic_vector(0 to 63);
        key : IN  std_logic_vector(0 to 63);
        clk : IN  std_logic;
        rst : IN  std_logic;
        soc : IN  std_logic;
        encrypt : IN  std_logic;
        dataOut : OUT  std_logic_vector(0 to 63);
        busy : OUT  std_logic;
        dataReady : OUT  std_logic
       );
   END COMPONENT;


  --Inputs
  signal dataIn : std_logic_vector(0 to 63) := (others => '0');
  signal key : std_logic_vector(0 to 63) := (others => '0');
  signal clk : std_logic := '0';
  signal rst : std_logic := '0';
  signal soc : std_logic := '0';
  signal encrypt : std_logic := '0';

      --Outputs
  signal dataOut : std_logic_vector(0 to 63);
  signal busy : std_logic;
  signal dataReady : std_logic;

  -- Clock period definitions
  constant clk_period : time := 1 ns;
```

```
BEGIN

      -- Instantiate the Unit Under Test (UUT)
  uut: DES_TOP PORT MAP (
        dataIn => dataIn,
        key => key,
        clk => clk,
        rst => rst,
        soc => soc,
        encrypt => encrypt,
        dataOut => dataOut,
        busy => busy,
        dataReady => dataReady
      );

  -- Clock process definitions
  clk_process :process
  begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
  end process;
```

```
-- Stimulus process
stim_proc: process
begin
        --encryption
  key <= x"1046913489980131";
        dataIn <= x"0000000000000000";
        encrypt <= '1';

        rst <= '1';
  wait for 2 ns;

        rst <= '0';
        soc <= '1';
  wait for clk_period*10;
        soc <= '0';
        encrypt <= '0';
        wait for clk_period*7;

        --decryption
        dataIn <= dataOut;
        soc <= '1';
        wait for clk_period*10;
        soc <= '0';

        wait for clk_period*10;



    wait;
  end process;

END;
```

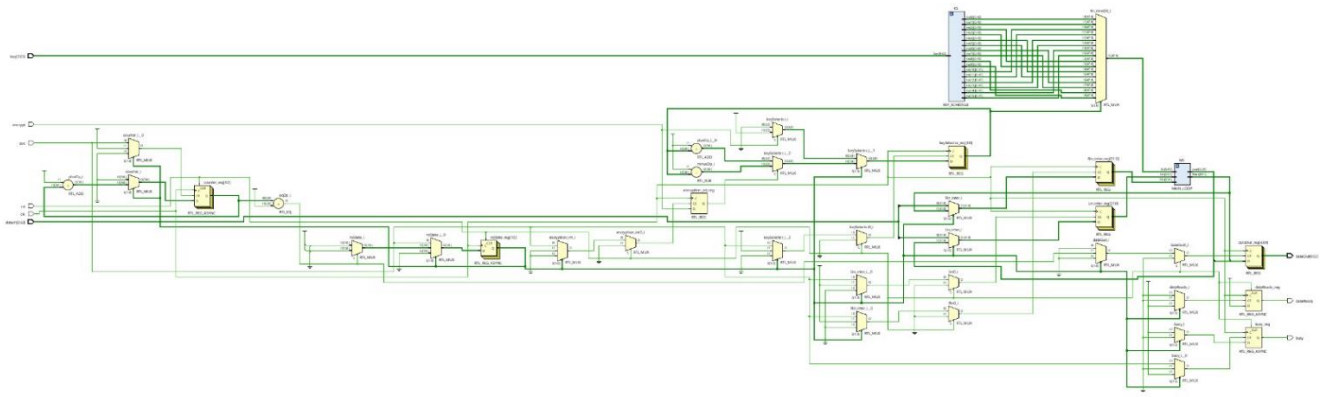
Output with Trojan :

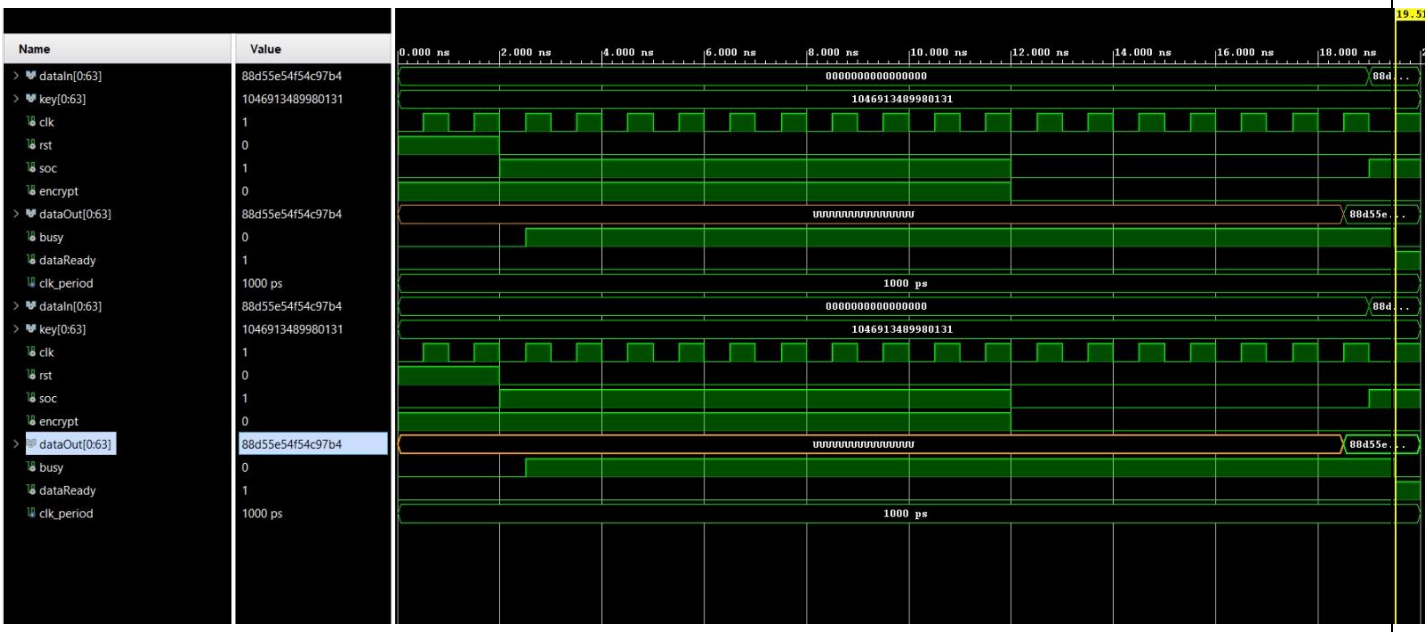




The figure above shows the simulation results for a Des implementation with the activated trojan giving the User an incorrect output while leaking the correct output through a signal that the attacker can place and route in a hidden manner.

Without trojan





Power Analysis:

- We decided to simulate the Des implementation with and without trojan on the Xilinx Zynq-7000 to get an understanding the overhead that is generated with Trojan insertion.
- Both designs were run through a post-implementation power analysis.

1. Summary

Total On-Chip Power (W)	7.640
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	7.479
Device Static (W)	0.160
Effective TJA (C/W)	1.9
Max Ambient (C)	85.7
Junction Temperature (C)	39.3
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

2.1 Environment

Ambient Temp (C)	25.0
ThetaJA (C/W)	1.9
Airflow (LFM)	250
Heat Sink	medium (Medium Profile)
ThetaSA (C/W)	3.4
Board Selection	medium (10"x10")
# of Board Layers	12to15 (12 to 15 Layers)
Board Temperature (C)	25.0

2.2 Clock Constraints

Clock	Domain	Constraint (ns)

3. Detailed Reports

3.1 By Hierarchy

Name	Power (W)
DES_TOP	7.479
trojan_counter	0.763

- Hence we can clearly see that the post-implementation power analysis shows a higher power use, these values can give us a general idea of how trojan addition can change power consumption
- These values only provide a general idea, a real life trojan insertion attempt would optimize placement and routing to ensure only a minor increase to power consumption.

1. Summary

Total On-Chip Power (W)	6.802
Design Power Budget (W)	Unspecified*
Power Budget Margin (W)	NA
Dynamic (W)	6.647
Device Static (W)	0.155
Effective TJA (C/W)	1.9
Max Ambient (C)	87.2
Junction Temperature (C)	37.8
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

2.1 Environment

Ambient Temp (C)	25.0
ThetaJA (C/W)	1.9
Airflow (LFM)	250
Heat Sink	medium (Medium Profile)
ThetaSA (C/W)	3.4
Board Selection	medium (10"x10")
# of Board Layers	12to15 (12 to 15 Layers)
Board Temperature (C)	25.0

2.2 Clock Constraints

Clock	Domain	Constraint (ns)

3. Detailed Reports

3.1 By Hierarchy

Name	Power (W)
DES_TOP	6.647

EGCP 447 TERM PAPER

Teamwork Contribution Sheet

I understand that providing false information could result in me failing the course. Preparer's Signature (sign the initial):	
Team Members' Name	Kaavya varshitha raman shantha
Description of Job	Running and simulating an implementation of DES and analyzing design for trojan insertion. Introduction and other sections of report
Team Members' Name	Ashwin Koshy John
Description of Job	Implementing trojan design and simulation section of project report
Team Members' Name	Reddy Sowmya Vangumalla
Description of Job	Testing Inserted trojan and analyzing it's effect. Body of project report.