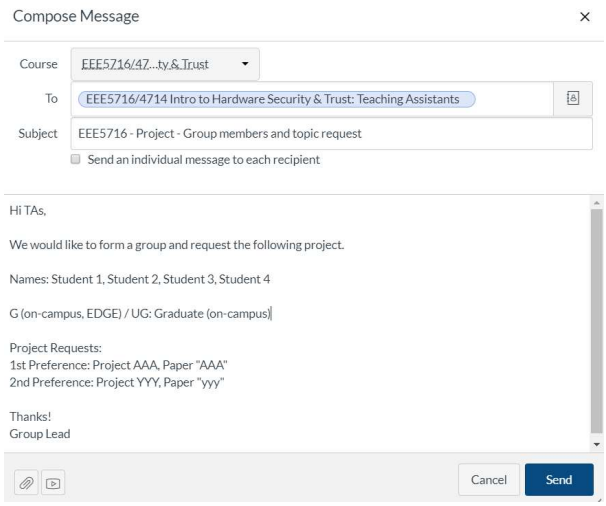
Note:
Some projects found in this file may only be applicable to on-campus students, this will be specified on the page. Search for "UG/Undergrad" or "G/Grad" to see if there are any differences between the two sections.

Deadlines:
- **6 February 2020 11:59pm:** Project request submission due to TAs. (Submit through Canvas per instructions in lecture.)
- **23 April 2020 11:59pm**: Final project deliverables and report due (Submit in Canvas.)
- **During Finals Week:** TAs may request a project demonstration by appointment, depending on the project and TAs ability to replicate results.

NOTE: The deadlines and deliverables may change due to unforeseen reasons. Please follow CANVAS announcements and discussion thread for further updates.

***After* projects are announced:**
Submit your request through Canvas (All TAs, CC your group members)
One submission for entire Group
Include:
Names:
Full name of your members
G / UG:
Graduate (On-campus, EDGE) or Undergraduate
Project Requests:
1st and 2nd project preferences



Important Points:

- Hardware (FPGA, Microcontroller, USB O'scope, etc.) must be purchased by the group.
- Script-based deliverables will be graded as a part of your report.
- SCAN Lab access will be granted by the TAs, by request only.
- The ECE Linux server can provide most of the software tools you need for these projects (i.e. Design Compiler, IFV, Jasper), or will be free to download (i.e. Xilinx Vivado)
- **ALL CUSTOM SCRIPTS MUST HAVE COMMENTS FOR UNDERSTANDING**

Revision Notes:

- Jan 28, 2020 - Initial Release

Project No.: 1
Title: Implementing Ring Oscillator Physical Unclonable Function (RO-PUF)

Main Objective: In this project, you are required to implement a ring oscillator (RO) PUF **(UG:32-bit, G:64-bit)** in an FPGA.

Background: Physical unclonable function (PUF) is an emerging potential security block for generating volatile secret keys in cryptographic applications. There are different types of PUFs implemented in FPGA [1-2]. A PUF is described as unclonable due to its uniqueness being derived from the uncontrollable variations introduced during the manufacturing process. PUFs offer a high level of protection in cryptographic applications with strong volatile key storage. PUFs are issued a challenge and (ideally) produce a unique and reliable response in return. Since this response is unique to the device, it can therefore be used as a device ID or key.

For FPGA implementations, ensure the routing is nearly identical between ring oscillator blocks. This is crucial since a RTL-based design without any routing constraints allows the synthesis and layout tool to provide an optimized structure which typically is not symmetric and identical. Hence, for most FPGAs, one needs to incorporate all possible routing constraints and design and place the hard-macro for symmetric design.

Project Goal:
The students are expected to design a **(UG:32-bit, G:64-bit)** RO-PUF Architecture and evaluate the quality of the designed PUF using the metrics – reliability, uniqueness, and uniformity/randomness.

Software Requirements:
Xilinx ISE or Vivado – For PUF implementation on Xilinx FPGAs
MATLAB, Octave, Python, R, C/C++ etc. for performance analysis.

Hardware Requirements:
We recommend that students use the FPGA kit that was suggested during lecture for PUF implementation.

Report to submit:
1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation, and problem statement. **(10%)**
   - RO-PUFs Implementations (**25%**)
   - Result and Effectiveness (**30%**)
   - Conclusion & Personal comments **(10%)**
   - All codes (append separately at the end of main report, no need to maintain IEEE format) **(20%)**
   - Novelty, clarity, organization, etc. **(5%)**

Submission guideline:
1. Submit .zip file with name: Project#Group#
2. Include all the following:
   a. Report_Project#Group#.pdf
   b. Run_Project#Group# (Folder, this will be working directory for running your code)
      i. readme.txt (provide necessary instruction for running your code)
      ii. All codes

References:
[1] M. Majzoobi, A. Kharaya, F. Koushanfar and S. Devadas , "Automated design, implementation, and evaluation of arbiter-based PUF on FPGA using programmable delay lines" , 2014.
[2] A. Maiti et al., "A systematic method to evaluate and compare the performance of physical unclonable functions," In Embedded Systems Design with FPGAs, P. Athanas, D. Pnevmatikatos, and N. Sklavos (Eds.). Springer, New York, 245267.

Project No.: 2
Title: Implementing Arbiter Physical Unclonable Function (Arbiter PUF)

Main Objective: In this project, you are required to implement **(UG: One, G: Two)** delay-based strong PUFs – Arbiter PUF, XOR-Arbiter PUF, Feed-forward Arbiter PUF in FPGA.

Background: Physical unclonable function (PUF) is an emerging potential security block for generating volatile secret keys in cryptographic applications. There are different types of PUFs implemented in FPGA [1-2]. A PUF is described as unclonable due to its uniqueness being derived from the uncontrollable variations introduced during the manufacturing process. PUFs offer a high level of protection in cryptographic applications with strong volatile key storage. PUFs are issued a challenge and (ideally) produce a unique and reliable response in return. Since this response is unique to the device, it can therefore be used as a device ID or key.

The Delay-based arbiter PUF and its variants are very good choice for ASIC implementation. However, for FPGA implementation, they require some additional considerations. For delay-based PUFs, the primary requirement is to have all the delay paths and blocks to be identical in layout so that there is no systematic (and predictable) variation in the outcome due to the path mismatch. For FPGA, this is crucial since a RTL-based design without any routing constraints allows the synthesis and layout tool to provide an optimized structure which typically is not symmetric and identical. Hence, for most FPGAs, one needs to incorporate all possible routing constraints and design and place the hard-macro for symmetric design.

Project Goal:
The students are expected to design **(UG: One, G: Two)** 64-bit each Arbiter, XOR-Arbiter and Feed-Forward Arbiter Architectures and evaluate the quality of the designed PUF using the metrics – reliability, uniqueness, and uniformity/randomness.

Software Requirements:
Xilinx ISE or Vivado – For PUF implementation on Xilinx FPGAs
MATLAB, Octave, Python, R, C/C++ etc. for performance analysis.

Hardware Requirements:
We recommend that students use the FPGA kit that was suggested during lecture for PUF implementation.

Report to submit:

3. IEEE conference format. (4-6 pages)
4. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation, and problem statement. **(10%)**
   - UG:2 - G:3 PUF Implementations **(25%)**
     - Architecture and principle
     - Challenge-response pair technique
     - Algorithms for post-processing if necessary
   - Result and Effectiveness (**30%**)
   - Conclusion & Personal comments **(10%)**
   - All codes (append separately at the end of main report, no need to maintain IEEE format) **(20%)**
   - Novelty, clarity, organization, etc. **(5%**)

Submission guideline:

3. Submit .zip file with name: Project#Group#
4. Include all the following:
   a. Report_Project#Group#.pdf
   b. Run_Project#Group# (Folder, this will be working directory for running your code)
      i. readme.txt (provide necessary instruction for running your code)
      ii. All codes

References:

[1] M. Majzoobi, A. Kharaya, F. Koushanfar and S. Devadas , "Automated design, implementation, and evaluation of arbiter-based PUF on FPGA using programmable delay lines" , 2014.
[2] A. Maiti et al., “A systematic method to evaluate and compare the performance of physical unclonable functions,” In Embedded Systems Design with FPGAs, P. Athanas, D. Pnevmatikatos, and N. Sklavos (Eds.). Springer, New York, 245267.

Project No.: 3
Title: Hardware Trojan Automated Insertion Tool

Main Objective:
In this project, you are required to design an automated tool that will insert a Hardware Trojan into a design to perform unwanted activity.

Background:
Due to the globalization of the semiconductor design and fabrication process, ICs are vulnerable to malicious modifications, referred to as hardware Trojans [1]. These hardware Trojans can create backdoors in the design through which sensitive information can be leaked and other possible attacks (e.g., denial of service, reduction in reliability, etc.) can be performed. Trojan circuits, by design, are typically activated under very specific conditions (e.g., connected to low-transition probability nets or sensing a specific design signal such as power or temperature), which makes them unlikely to be activated and detected using random or functional stimuli. The Trojan circuit is primarily composed of two components; a component responsible for activation mechanisms (referred to as triggers) and other component responsible for producing the malicious effect (referred to as payloads).

Software Requirements:
Xilinx ISE or Vivado (Synopsys VCS is also acceptable).

Overall Requirements:
For this project, you need to design a tool that will read a design file and automatically insert Trojan into the design. The requirements of the tool are given below,

- The Trojan insertion needs to be implemented in the RTL or gate-level.
- The Trojan needs to be triggered under very specific/rare condition. The triggering circuit needs to have following requirements:
  1. The rareness of the triggering condition needs to be taken as an input from the user. You are free to choose a metric which quantitatively defines the rareness of the triggering condition. Examples of such metrics in RTL level is statement hardness [1] and in gate level is transition probability [3].
  2. To meet the given rareness requirement, the tool need to automatically insert required circuitry in the design. For example, in the gate level the tool first needs to identify rare nets in the design and then add necessary circuitry which will activate the Trojan trigger when those rare nets are simultaneously activated.
- The tool needs to report the input stimulus that trigger the Trojan to the user.
- The tool needs to automatically insert the payload circuitry. The payload circuit needs to have the following requirements:
  1. The user will give a list of input pins and the tool will insert the payload that will propagate the logical values of the input pins to output pins when the triggering input stimulus is applied. For example, in an encryption module, the user will specify the name of the key input pins and the tool will generate the payload circuit to propagate key value to primary output once the triggering input stimulus is applied.
  2. The user will give an output pin name and the desired output value and the tool needs to insert a payload that will make the given output pin to that desired logical value once the triggering input stimulus is applied.
- Demonstration:
  1. You need to show the input triggering stimulus generated by the tool.
  2. You need to apply a large number of random patterns and show that the Trojan is not activated.
  3. You need to demonstrate that the Trojan is activated given the triggering stimulus and show that once activated the payload is performing its desired function.

Report to submit:

1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation, and problem statement. **(10%)**
   - Background and Prior techniques **(5%)**
   - Techniques for Inserting Trojans **(25%)**
     - Necessary algorithms

- Design flow of tools
- Results (Implementation **15%** + Effectiveness **10%**)
- Conclusion & Personal comments (**10%**)
- All codes (append separately at the end of main report, no need to maintain IEEE format) (**20%**)
- Novelty, clarity, organization, etc. (**5%**)

Submission guideline:

1. Submit .zip/.rar file with name: Project#Group#
2. Include all the following:
   - **a.** Report_Project#Group#.pdf
   - **b.** Run_Project#Group# (Folder, this will be working directory for running your code)
     - **i.** readme.txt (provide necessary instruction for running your code)
     - **ii.** All codes

[1] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," IEEE Design and Test of Computers, 2010.
[2] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level," IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 190-195, 2013.
[3] H. Salmani, R. Karri, and M. Tehranipoor, "On design vulnerability analysis and trust benchmarks development," Proceedings of IEEE 31st International Conference on Computer Design (ICCD), pp. 471-474, 2013.

Project No.: 4
Title: Machine Learning Attack on Arbiter PUF

Main Objective:
In this project, you are required to perform machine learning (ML) based attack on arbiter PUFs.

Background:
Research shows that arbiter-PUFs are vulnerable to certain machine learning based modeling attacks (please see **Ref. [1], [2]**). Here adversaries try to make a black-box predictive (mathematical) model, as accurate as possible, of the arbiter-PUF. This model behaves as a software clone of the physical system, and produces probable Challenge/Response Pair (CRP) set without requiring access to the PUF. The main purpose of such attack is to generate mimicked response with the mathematical model for a given challenge, where the model is first developed using original CRPs from the physical system.

In the simplest form, such a machine learning based modeling attacks needs the following:

1. Training up an adaptive system (based on machine learning algorithm) with a sufficient size of CRP set that has already been acquired from the original PUF.
2. Feeding new input (i.e. challenge set) to the trained system (which now works as the software clone of the original PUF) to generate response and evaluate the effectiveness by comparing the result with that from the original PUF.
3. Improve the efficiency of the attack by reducing required size of training set for a target effectiveness/error.

Project Goal:

1. Create a mathematical model of an Arbiter PUF [1-3]. Model the relationship between challenge and response pairs. The overall delays of the signals are modeled as the sum of the delays in the stages. In this model, one can express the final delay difference Δ between the upper and the lower path in an Arbiter PUF **(Both UG and G)**.
2. Create synthetic challenge-response pairs (CRPs) for the PUF. Plot the histogram of the Hamming Weight of the responses for the PUF **(Both UG and G).**
3. Perform a machine learning attack on the arbiter synthetic PUF **(Both UG and G).**
4. Introduce measurement noise (5%) by randomly flipping the bits of the responses. Perform the same machine learning attack **(only G).**
5. Create a 2-XOR arbiter PUF and perform the same machine learning attack **(only G).**
6. For the given CRP dataset, implement **(UG: One, G: One)** machine learning technique for making a successful attack.
7. Use three training set sizes (total 12000 challenges/samples available)
   - 2000, 6000, 10000 samples
   - All remaining samples are used for evaluation
8. Target maximum accuracy for each sample size.
9. Target lowest training size for 80% accuracy.
10. Evaluate your model. Run your attack model and collect results (20 trials) for each training size.

For each training size, choose the samples randomly and make 20 trials. Report the average accuracy result (i.e. %of successful detection).

Software Requirements:
Any software and scripting language capable of performing numerical analysis and building machine learning-based trainers and classifiers (MATLAB, R, Python, C/C++).

Dataset Description:
CRPSets.zip contains (12000x65) data for a 64-stage arbiter-PUF with 12000 challenges, as indicated by rows. Each challenge is 64bit long (Column1-64), and the corresponding response is one bit – either 0/1, as given by column 65.

Results to submit:

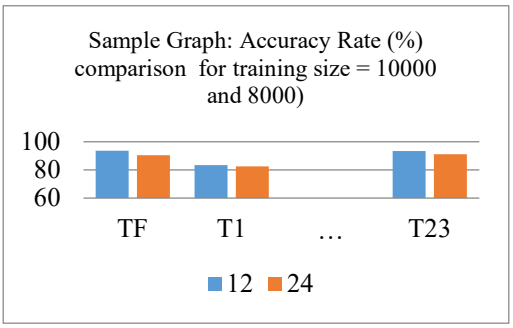
1. Describe how you created the mathematical model. Mention the distribution of delay that you chose for the PUF modeling.

2. Compare your results with [1,2].
3. For each attack model, you must provide the followings:
   - Working flow-chart/algorithm
   - Used boundary conditions, parameters, etc. as necessary.
4. For each training cases, you must provide the followings:
   - Average accuracy rates over the 10 trials (figures, tables, etc.).
   - Average training time (if appropriate) and evaluation time (if appropriate).
   - Comparative results
     - Among different sample size
     - Among different attack model (if applicable)
     - You should also provide any other necessary data to justify your implementation.
   - Accuracy rate vs. training sample required: Table and Plot

Some sample tables/plots are given for your convenience.

Sample Table: Accuracy rate (%) for Training Size =10000 (Random Trial)

Sample Table: Accuracy rate (%) for Training Size =10000 (Over 10 Trials)	
Avg	
SD	
Max	
Min	



Report to submit:

1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation and problem statement. **(10%)**
   - Prior techniques **(5%)**
   - Model selection and design **(25%)**
     - Reason of choice, Brief description, Pros/Cons,
     - Short workflow/algorithm, Settings/ architecture, as necessary.
   - Model evaluation results **(Implementation 15% + Effectiveness 10%)**
   - Conclusion & Personal comments **(10%)**
   - All codes (append separately at the end of main report, no need to maintain IEEE format) **(20%)**
   - (Novelty, clarity, organization, etc. **(5%)**

Submission guideline:

1. Submit .zip file with name: Project#Group#
2. Include all the following:
   a. Report_Project#Group#.pdf
   b. Run_Project#Group# (Folder, this will be working directory for running your code)
      i. readme.txt (provide necessary instruction for running your code)
      ii. All codes
      iii. Used data set

References:

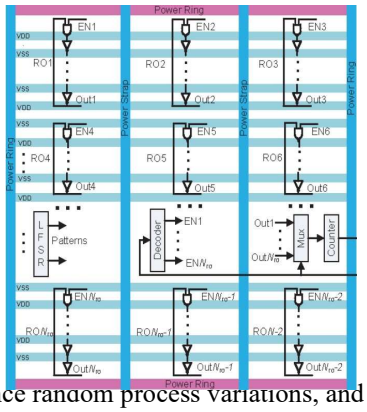
[1] Rührmair, Ulrich, et al. "PUF modeling attacks on simulated and silicon data." IEEE Transactions on Information Forensics and Security 8.11 (2013): 1876-1891.
[2] Rührmair, Ulrich, et al. "Modeling attacks on physical unclonable functions." Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010.
[3] Lim, Daihyun. “Extracting secret keys from integrated circuits”. Diss. Massachusetts Institute of Technology, 2004.

Project No.: 5
Title: Machine Learning-based Classifier Implementation for Hardware Trojan Detection

Main Objective:
In this project, you are required to design machine learning (ML) based classifiers to detect a hardware Trojan.

Background:
A hardware Trojan, in its simplest form, can be viewed as an unauthorized design that is inserted by adversaries without the designers' knowledge to the original chip, in order to trigger specific actions. Detecting such Trojans in a fabricated chip is extremely difficult, since the designer does not have any knowledge of it (e.g. whether the chip is actually Trojan-free or not; the size, type, and location of the Trojan (if inserted); behavior of the inserted Trojan; activity, and so on). An interesting power-based detection mechanism is to monitor power supply variations (or some other form that may infer such variations) for any suspicious activity that may arise from inserted Trojan(s).

In **Ref. [1]**, authors put identical ring oscillators in different locations of a fabricated chip to monitor the power supply network activity by measuring the frequencies of the ROs. The key idea behind is that RO frequencies are affected by power supply and temperature variations (runtime activity related), and capacitance, threshold voltage, etc. variations (from manufacturing process). Usually, Trojan switching activity draws current through power network (resistor) → Causes IR Drop → Causes Frequency Drop in ROs; hence ROs in Trojan-inserted chips will run slower than Trojan-free (Golden) chip when Trojan is activated in an ideal scenario! One can use this idea for detecting hardware Trojans.



However, the real-life scenario is much more complicated. There is no one value of frequency that works as a pass/fail threshold for detection. ROs experience random process variations, and therefore produce slightly different frequencies even with similar design and operating conditions (recall the RO frequency data in homework 2), even for golden (i.e. Trojan-free) chips. Moreover, process variation for the given samples may overshadow the effects coming from activated Trojans. And RO-frequencies not only depend on only presence of a Trojan, but also on the type (combinational or sequential), size (small or large), and location (near or far of a RO) of the Trojans. It requires 'smart' classifiers that can still differentiate among RO-frequencies incorporating all these real-life noises. That is why you need to **design a classifier using appropriate machine learning algorithms that can take chip RO frequencies and declare whether it is Trojan inserted or not.**

Project Goal:

1. Implement the following **(UG: One, G: Two)** cases:
   - **Case 1.** You have some known samples of both Golden chips and Trojan inserted chips. (i.e. RO data from both type of chips are known), and both can be used for training the classifier. (However, the type of the Trojan is unknown.)
   - **Case 2.** You only have some known samples of Golden chips, i.e. only golden data can be used for training the classifier.
   - **Case 3.** You have samples that are completely unidentified (no knowledge about the samples whether they are golden, Trojan-inserted, or a mixture of both) to train the classifiers.
2. Implement **(UG: One, G: Two)** different classification techniques of your choice.
3. Evaluate your classifier accuracy with given sample size (33 total samples available):
   - 6 samples (Case 1: 3TF and 3TI; Case 2: All 6TF; Case 3: All 6 Unknown)
   - 12 samples (Case 1: 6TF and 6TI; Case 2: All 12TF; Case 3: All 12 Unknown)
   - 24 samples (Case 1: 12TF and 12TI; Case 2: All 24TF; Case 3: All 24 Unknown)
   - All remaining samples are used for evaluation

For each case, choose the samples for the training set randomly (20 trials) and use the remaining chips/samples for evaluation. Run your classification/detection and collect results (20 trials) for each Trojan type (23 total), and report the average accuracy result (i.e. %of successful detection).

Software Requirements:

Any software and scripting language capable of performing numerical analysis and building machine learning-based classifiers (MATLAB, R, Python, C/C++).

Dataset Description:
ROFreq.zip contains 33 spreadsheets (Chip1, Chip2, …, Chip33) that refer data from 33chips (samples). Each spreadsheet has following specs:

- Each row indicates Trojan Free/ Trojan-inserted data
  - Trojan Free (golden) data => Row (1 & 25)
  - Trojan-inserted data => Row 2 to Row 24.
- Each column indicates frequency of RO# in the network (RO1-RO8).

For details, check Ref. [1].

Results to submit:

1. For each classifier, provide the followings:
   - Working flow-chart/algorithm and used boundary conditions, parameters, etc. as necessary.
2. For each problem case, provide the followings:
   - Average true positive and false positive rates over the 20 trials (figures, tables, etc.).
   - Average training time (if appropriate) and evaluation time (if appropriate).
   - Comparative results
     - Among different sample size
     - Among different cases (if applicable)
   - You can also provide any other necessary data to justify your implementation.

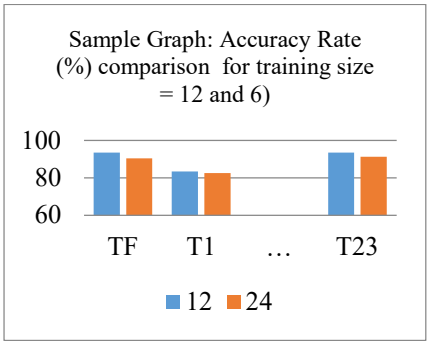
Some sample tables/plots are given for your convenience.

Sample Table: Accuracy rate (%) for Training Size =12 (Random Trial)

TF	T1	T2	…	T23
-	-	-	…	-

Sample Table: Accuracy rate (%) for Training Size =12 (Over 20 Trials)

	TF	T1	T2	…	T23
Avg	-	-	-	…	-
SD					
Max					
Min					



Report to submit:

1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation and problem statement. **(10%)**
   - Prior techniques **(5%)**
   - Classifiers selection and design **(25%)**
     - Reason of choice, Brief description, Pros/Cons,
     - Short workflow/algorithm, Settings/ architecture, as necessary.
   - Classifier evaluation results **(Implementation 15% + Effectiveness 10%)**
   - Conclusion & Personal comments **(10%)**
   - All codes (append separately at the end of main report, no need to maintain IEEE format) **(20%)**
   - (Novelty, clarity, organization, etc. **(5%)**

Submission guideline:

1. Submit .zip/.rar file with name: Project#Group#
2. Include all the following:

- **a.** Report_Project#Group#.pdf
- **b.** Run_Project#Group# (Folder, this will be working directory for running your code)
  - **i.** readme.txt (provide necessary instruction for running your code)
  - **ii.** All codes
  - **iii.** Used data set

References:

[1] Shane Kelly, Xuehui Zhang, Mohammed Tehranipoor, and Andrew Ferraiuolo, "Detecting Hardware Trojans using On-chip Sensors in an ASIC Design." Journal of Electronic Testing 31, no. 1 (2015): 11-26.

Project No.: 6
Title: Design and Security verification of a First-in-First-out (FIFO).

Background: FIFOs are commonly used buffering and flow control hardware. It is used for lots of applications such as interface between software-hardware or two different clock domains. The FIFO maintains the data sequence where data written first is read out first. The FIFO should maintain the data integrity by avoiding overflow (data lost when FIFO is full) and underflow (reading from empty FIFO)

Students are required to design 8-bit(UG)/16-bit(G) FIFO with depth 4(UG)/16(G) and then verify it using dynamic and formal approaches. Then write a test bench to verify the FIFO design implementation. In case of found errors, go back and fix your FIFO implementation. After going back and forth and satisfied with your implementation, make no further changes to your design and testbench.

After that list down 5 (UG), 15(G) specifications of your FIFO in natural language which you think your design should obey all the time. Covert these natural language specifications to formal properties using System Verilog Assertions (SVA). Give your FIFO implementation and the SVA properties to the JasperGold tool to formally verify the FIFO implementation. Note which properties fail and record the counterexamples generated by the tool.

Software Requirements:
Any Verilog/VHDL/System Verilog simulator (Vivado Preferred).
JasperGold for formal verification (Available on ECE server)

Hardware Requirements:
No Hardware Requirements

Deliverables:

- FIFO implementation (FIFO.v)
- FIFO testbench (FIFO_tb.v)
- FIFO formal properties (FIFO_v.sv)
- Bind files and TCL script (To run automatically in Jaspergold)
- Traces screenshots and VCD dump files of counterexamples
- REPORT (Very Important)

Report to submit:

1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation, and problem statement. **(10%)**
   - FIFO Implementations (**25%**)
   - Result and Effectiveness (**30%**)
   - Conclusion & Personal comments **(10%)**
   - All codes (append separately at the end of main report, no need to maintain IEEE format) **(20%)**
   - Novelty, clarity, organization, etc. **(5%)**

Submission guideline:

1. Submit .zip file with name: Project#Group#
2. Include all the following:
   a. Report_Project#Group#.pdf
   b. Run_Project#Group# (Folder, this will be working directory for running your code)
      i. readme.txt (provide necessary instruction for running your code)
      ii. All codes

References:

- Cummings, Clifford E. "Simulation and synthesis techniques for asynchronous FIFO design." *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*. 2002.
- https://www.doulos.com/knowhow/sysverilog/tutorial/assertions/

Project No.: 7
Title: Side-Channel Attacks on Data Encryption Standard (DES)

Main Objective: In this project, you will extract the key of a Data Encryption Standard (DES) cryptographic algorithm through a side-channel attack using power analysis, with as few as possible collected power traces.

Background:
In cryptography, a side-channel attack is an attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms (contrast with cryptanalysis). For example, timing information, power consumption, electromagnetic emanations or even acoustic sound can provide an extra source of leaked information, which can be exploited to break the system. Some side-channel attacks require technical knowledge of the internal operation of the system on which the cryptography is implemented, although others such as differential power analysis are effective as black-box attacks. Many powerful side-channel attacks are based on statistical methods pioneered by Paul Kocher.

Power analysis is a form of side-channel attack in which the attacker studies the power consumption of a cryptographic hardware device (e.g., smart card, tamper-resistant "black box", or integrated circuit). The attack can non-invasively extract cryptographic keys and other secret information from the device. Simple power analysis (SPA) involves visually interpreting power traces, or graphs of electrical activity over time. Differential power analysis (DPA) is a more advanced form of power analysis which can allow an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations.

Project Goal:
The students are expected to mount a DPA attack on the *public* power traces provided by http://www.dpacontest.org/tables.php. The secret key for the DES should be found by the attack. Students should optimize their algorithm so that it can successfully obtain the key with as few as possible power traces.

Overall Requirements:
For this project, you need to implement a DPA on power traces that were collected from DES. You can download these traces from the public database http://www.dpacontest.org/tables.php

1. Extract each bit of the key from your attack and show your detailed procedure.
2. Attempt to optimize your code so that the key can be found with as few as possible power traces.

Demonstration Requirements:

1. You need to run your code and get the figures of all the possible guesses for one byte of the round-key, in a short period of time.

2. Compare the figures of the right guess with all the other wrong guesses. The right guess should show apparent difference with the wrong guesses.

3. Show how to get the key with the round-key.

Software Requirements:
Recommended programming languages include (but not limited to): MATLAB, C/C++, C#, Python.

Hardware Requirements:
None

Report to submit:

1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation, and problem statement. **(10%)**
   - Your DPA Implementations (**45%**)

  - Describe your method
  - Provide your algorithms
- Result and Effectiveness (**15%**)
- Conclusion & Personal comments **(5%)**
- All source code, documented appropriately, (excluding power trace files from website). Append as separately at the end of main report, no need to maintain IEEE format for source code. **(20%)**
- Novelty, clarity, organization, etc. **(5%)**

Submission guideline:

1. Submit .zip/.rar file with name: Project#Group#

2. Include all the following:

a) Report_Project#Group#.pdf

b) Run_Project5#Group# (Folder, this will be working directory for running your code)

c) Readme.txt (provide necessary instructions for running your code)

d) All source code (excluding power trace files from website).

References:

[1] http://www.dpacontest.org
[2] Clavier, Christophe, et al. "Practical improvements of side-channel attacks on AES: feedback from the 2nd DPA contest." Journal of Cryptographic Engineering 4.4 (2014): 259-274.
[3] Kocher, Paul, Joshua Jaffe, and Benjamin Jun. "Differential power analysis." Annual International Cryptology Conference. Springer Berlin Heidelberg, 1999.
[4] Zhou, YongBin, and DengGuo Feng. "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing." IACR Cryptology ePrint Archive 2005 (2005): 388.
[5] Prouff, Emmanuel. "DPA attacks and S-boxes." International Workshop on Fast Software Encryption. Springer Berlin Heidelberg, 2005.
[6] Guilley, Sylvain, et al. "Silicon-level solutions to counteract passive and active attacks." Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on. IEEE, 2008.
[7] Guilley, Sylvain, et al. "Improving side-channel attacks by exploiting substitution boxes properties." International Conference on Boolean Functions: Cryptography and Applications (BFCA). 2007.
[8] Hnath, William. Differential power analysis side-channel attacks in cryptography. Diss. Worcester Polytechnic Institute, 2010.

Project No.: 8
Title: Modeling Risk of Counterfeit Integrated Circuits

Main Objective: In this project, you are required to model the risk created by counterfeit integrated circuits (IC) in the electronics supply chain.

Background: The globalized electronics supply chain is vulnerable to counterfeit integrated circuits (IC). In fact, almost all electronic components pass through numerous entities before reaching their final destination. This creates unknown risks of counterfeit ICs being installed into critical systems. It is true that there are many methods attempting to detect counterfeit ICs; however, methods to quantify the risk posed by counterfeit ICs are not as abundant. The need exists for data-driven, quantifiable methods to model the risk associated with counterfeit ICs.

Project Goal:
The students are expected to develop metrics for modeling the risk of the most common types counterfeit ICs in the supply chain (UG: Recycled, Remarked; G: Recycled, Remarked, Cloned). This should include metrics to calculate the confidence level of relative-risk (i.e., metrics-based, not a subjective approach).

The input data should be from internal hardware properties such as (e.g., PUF, CDIR, ECID, metering, watermarks, side-channels, etc.), rather than from external test equipment. For example, PUFs are often measured in terms of Hamming Distance, which can be a useful metric for intrinsic authentication. Methods may be developed from many areas including: testing and formal verification, game theory, predator-pray models, defender-attacker-defender sequential game models, optimization problems (knapsack), linear & dynamic programming, etc. The students may use open-source tools to aid in their methodology.

Software Requirements:
open-source tools such as Google OR-Tools, GLPK, CP-SAT, risk modeling tools, formal verification tools, etc. programming languages such as MATLAB, Python, C/C++, etc.

Hardware Requirements:
None.

Report to submit:
1. IEEE conference format. (4-6 pages)
2. Report should include (tentative marks dist.):
   - Top Sheet with group members name and ID
   - Introduction, motivation, and problem statement. **(10%)**
   - Risk Metrics and Implementations (**25%**)
   - Result and Effectiveness (**30%**)
   - Conclusion & Personal comments **(10%)**
   - All codes (append separately at the end of main report, no need to maintain IEEE format) **(20%)**
   - Novelty, clarity, organization, etc. **(5%)**

Submission guideline:
1. Submit .zip file with name: Project#Group#
2. Include all the following:
   a. Report_Project#Group#.pdf
   b. Run_Project#Group# (Folder, this will be working directory for running your code)
      i. readme.txt (provide necessary instruction for running your code)
      ii. All codes

References:

[1] U. Guin, D. Dimase, and M. Tehranipoor, "A comprehensive framework for counterfeit defect coverage analysis and detection assessment." Journal of Electronic Testing: Theory and Applications, vol. 30, no. 1, pp. 25–40, 2014.
[2] Z. Collier, D. DiMase, S. Walters, M. Tehranipoor, J. Lambert, and I. Linkov, "Cybersecurity standards: managing risk and creating resilience," IEEE Computer, vol. 47, no. 9, pp. 70–76, September 2014.

[3] J. Graf, "Trust games: How game theory can guide the development of hardware Trojan detection methods," 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) 2016, pp. 91-96.
[4] A. M. Smith, J. R. Mayo, V. Kammler, R. C. Armstrong and Y. Vorobeychik, "Using computational game theory to guide verification and security in hardware designs," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2017, pp. 110-115.
[5] W. Saad, A. Sanjab, Y. Wang, C. A. Kamhoua and K. A. Kwiat, "Hardware Trojan Detection Game: A Prospect-Theoretic Approach," in IEEE Transactions on Vehicular Technology, vol. 66, no. 9, pp. 7697-7710, Sept. 2017.
[6] M. Hristakeva, D. Shrestha, “Different Approaches to Solve the 0 / 1 Knapsack Problem,”, 2005
[7] S. Rahim, S. A. Khan, N. Javaid, N. Shaheen, Z. Iqbal, and G. Rehman. 2015. “Towards Multiple Knapsack Problem Approach for Home Energy Management in Smart Grid.” In Proceedings of the 2015 18th International Conference on Network-Based Information Systems (NBIS ’15). IEEE Computer Society, USA, 48–52
[8] Z. Collier, M. Hassler, J. Lambert, D. DiMase, and I. Linkov, Igor. “Supply Chains.”, (2019).