IIS5008
Hardware Security

PA1 Explanation

Andy, Yu-Guang Chen
Associate Professor, Department of EE
National Central University
andyygchen@ee.ncu.edu.tw
Slides Credit: TA Wei-Hung, Lin

Outline

- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

Outline

- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

What is a PUF

- Physical Unclonable Functions (PUFs)
  - Generates a **unique device fingerprint** based on the uncontrollable process variations
  - Highly resistant to duplication and unauthorized modifications
  - Applications of PUF
    - Device Authentication
    - Cryptographic Key Generation & Storage
    - Secure Communications & Encryption

CRP PUF

- Physical Unclonable Functions (PUFs)
  - In Challenge-Response Pair(CRP) protocols: R=f(C)
  - 3 fundamental properties
    - Uniqueness
    - Reliability
    - Randomness
  - Strong PUF (e.g., Arbiter PUF、Ring Oscillator PUF)

Ring Oscillator PUF

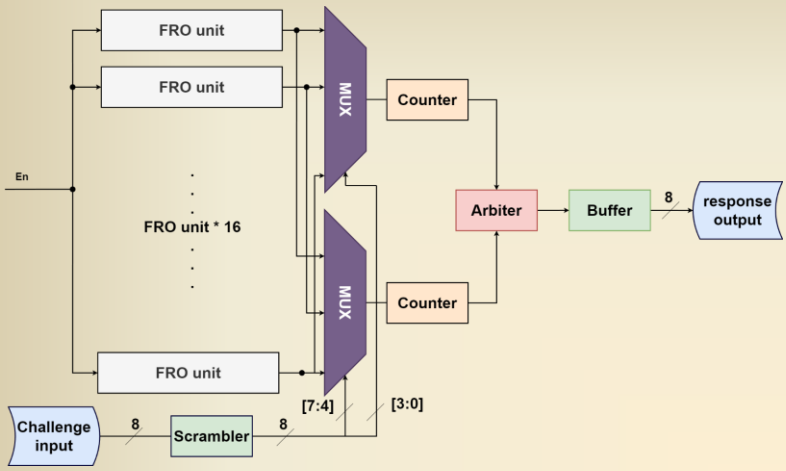
- Numerous ring oscillators (ROs) and takes into account their **relative frequencies**
- Each RO has slightly different oscillation frequencies because of random variations

```
localparam real DELAY_SETS [0:15][0:10] = '{
    '{0.11, 0.12, 0.13, 0.15, 0.27, 0.33, 0.46, 0.57, 0.62, 0.75, 0.88},
    '{0.14, 0.19, 0.22, 0.26, 0.31, 0.39, 0.41, 0.48, 0.52, 0.68, 0.84},
    '{0.10, 0.21, 0.23, 0.29, 0.35, 0.42, 0.49, 0.51, 0.65, 0.76, 0.83},
    '{0.16, 0.17, 0.18, 0.25, 0.28, 0.36, 0.45, 0.59, 0.62, 0.70, 0.82},
    '{0.11, 0.13, 0.29, 0.34, 0.38, 0.44, 0.53, 0.63, 0.69, 0.77, 0.90},
    '{0.12, 0.15, 0.28, 0.31, 0.37, 0.46, 0.55, 0.60, 0.66, 0.79, 0.86},
    '{0.18, 0.24, 0.26, 0.32, 0.40, 0.43, 0.50, 0.52, 0.61, 0.71, 0.87},
    '{0.10, 0.18, 0.22, 0.27, 0.30, 0.41, 0.46, 0.54, 0.64, 0.76, 0.89},
    '{0.11, 0.12, 0.16, 0.19, 0.36, 0.39, 0.47, 0.58, 0.67, 0.78, 0.83},
    '{0.12, 0.20, 0.25, 0.33, 0.38, 0.48, 0.56, 0.59, 0.65, 0.73, 0.81},
    '{0.14, 0.25, 0.29, 0.30, 0.41, 0.45, 0.53, 0.62, 0.69, 0.80, 0.90},
    '{0.13, 0.23, 0.27, 0.36, 0.38, 0.44, 0.52, 0.64, 0.75, 0.78, 0.89},
    '{0.10, 0.17, 0.26, 0.31, 0.42, 0.48, 0.50, 0.61, 0.68, 0.74, 0.82},
    '{0.15, 0.19, 0.22, 0.33, 0.37, 0.43, 0.51, 0.54, 0.63, 0.77, 0.84},
    '{0.11, 0.18, 0.24, 0.28, 0.42, 0.49, 0.55, 0.67, 0.70, 0.78, 0.85},
    '{0.10, 0.12, 0.17, 0.25, 0.29, 0.35, 0.46, 0.52, 0.66, 0.72, 0.88}
};
```

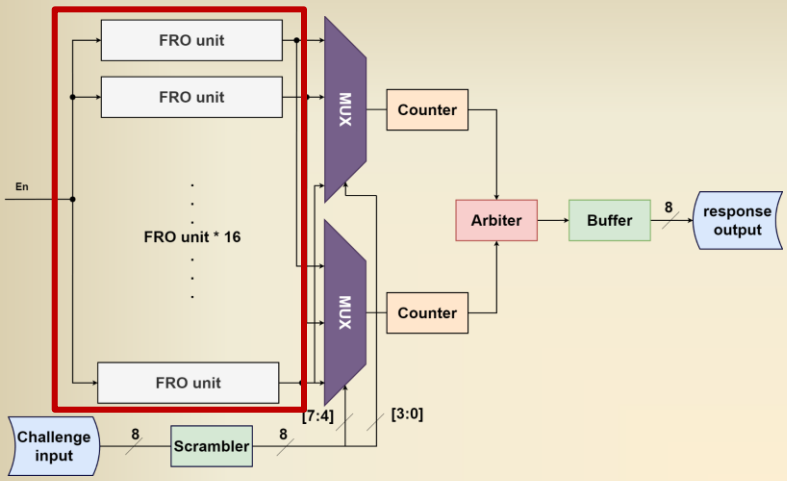
Outline

- Introduction & Background
- RO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

RO PUF Architecture

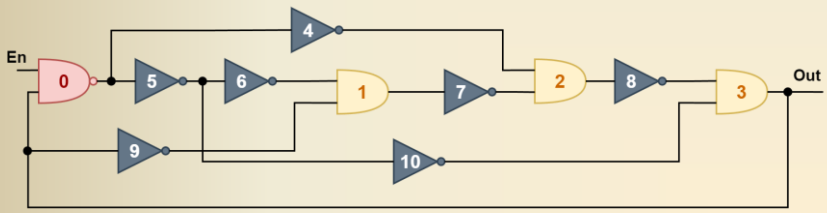


Feedforward Ring Oscillator

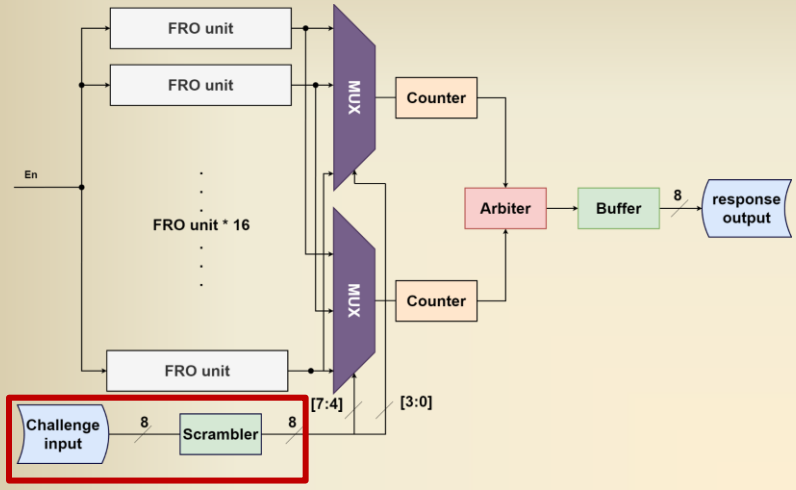


Feedforward Ring Oscillator

- Consists of 16 FROs for frequency variations
- Process variations will be simulated by assigning different delay
  - nand #(1.5) (nand_out, en, out);

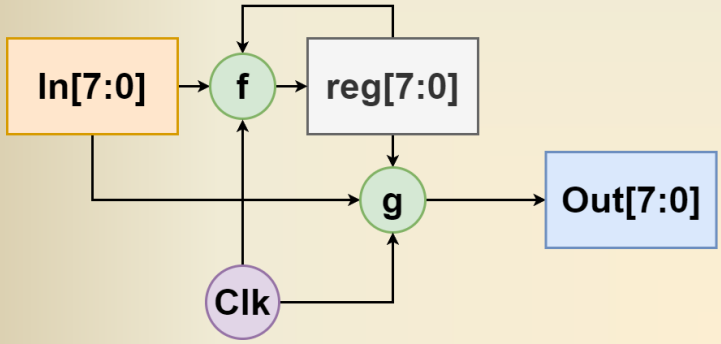


Scrambler



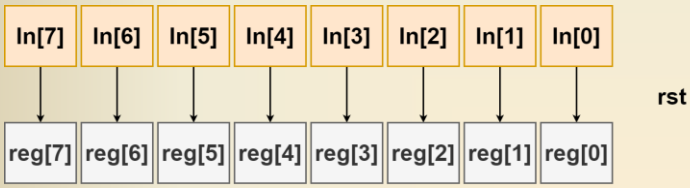
Scrambler

- Creates pseudo-random selection signals for MUXs
- 8-bit challenge input & 8-bit response



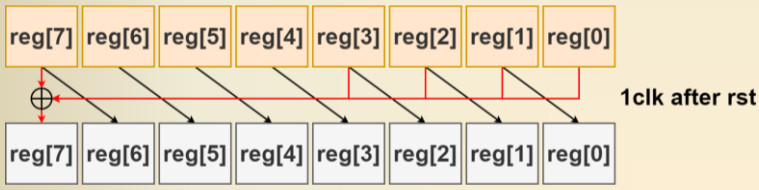
Scrambler

- Function f
  - A Linear Feedback Shift Register (LFSR)
  - The register is reset by input



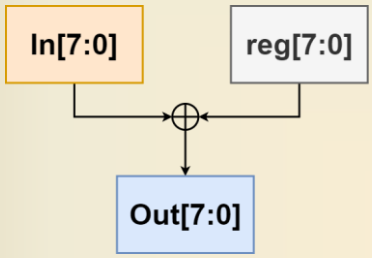
Scrambler

- Function f
  - The register is updated at each clock positive edge
  - x^7 + x^3 + x^2+ x^1 + 1 in GF(2^8)
  - **It will not generate 8'b0, and input will not test 8'b0, either**

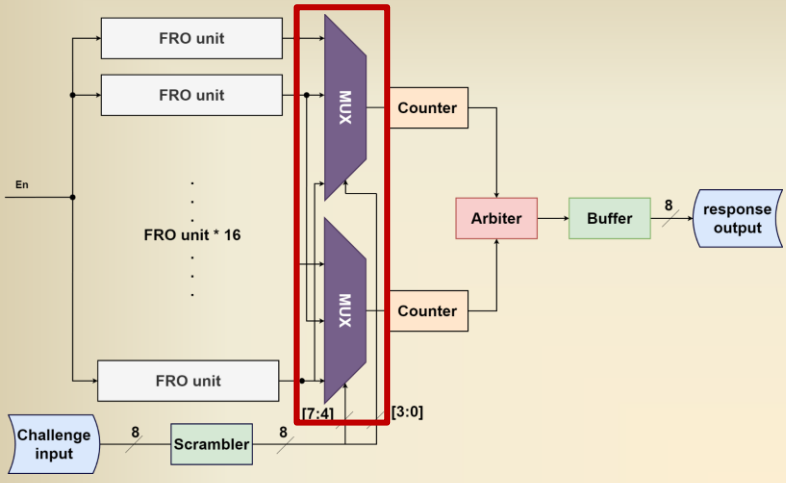


Scrambler

- Function g
  - A simple XOR to improve randomness



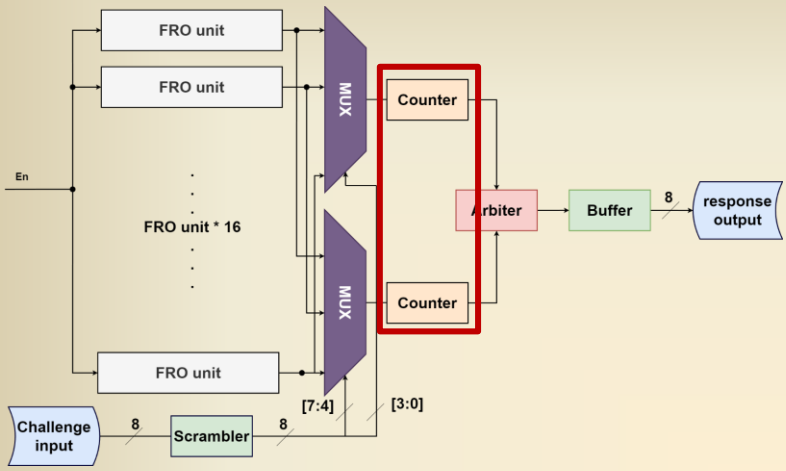
16 to 1 Multiplexer



16 to 1 Multiplexer

- Selects a candidate from 16 FROs
- Select signal generated by the Scrambler module

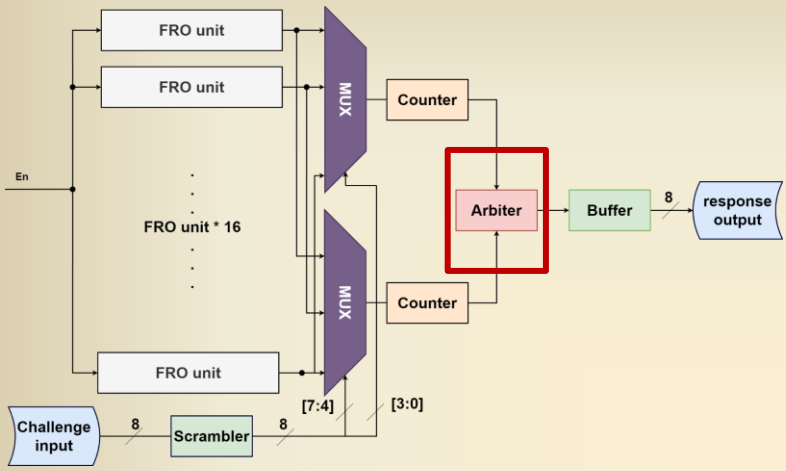
Counter



Counter

- Counts up to an input value and signals when it reaches the target
- Count value is determined by the parameter WIDTH
- The finish signal is set to 1 when the count reaches **2^WIDTH - 1**, otherwise it remains 0.
  - For WIDTH = 3(default), output 1 when reaches 7

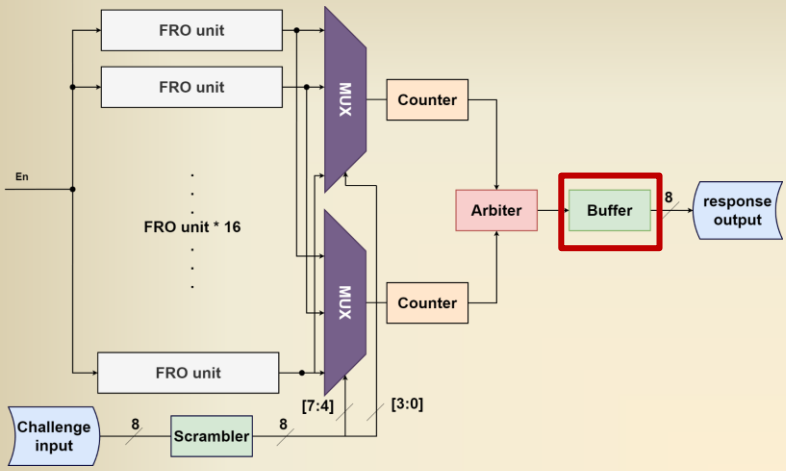
Race Arbiter



Race Arbiter

- Determines which signal finished first
  - Group A completes first, output = 1
  - Group B completes first, output = 0
  - They complete at the same time, output = 1

Buffer



Buffer

- Stores each bit from each race and outputs an 8-bit response
  - Buffer reset logic is for Arbiter and Counter
    - When system reset, buf_rst = 1
    - When Arbiter output valid signal, buf_rst = 1
    - Otherwise it remains 0.
  - Ready logic sets 1 when output signal is valid

Outline

- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

Top.v

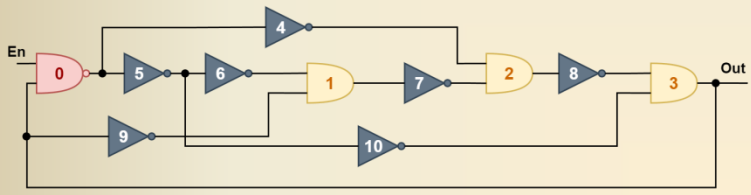
- DELAY_SETS will pass into each FRO
  - [0:15] means 16 FROs
  - [0:10] means 11 gates in each FRO
- WIDTH will pass into each Counter

```
module Top#(
    parameter real DELAY_SETS [0:15][0:10] = '{16{ '{11{0.1}}}},
    parameter WIDTH = 3
) (
    input clk,
    input en,
    input sys_rst_pos,
    input [7:0] chall_in,
    output [7:0] response,
    output ready
);
```


FRO.v

- DELAY_APR must be follow the order of gates as shown in the diagram
  - Nand gate is passed by DELAY_APR[0]

```
module FRO #(parameter real DELAY_ARR [0:10] = '{
        1.1, 1.2, 1.3, 1.4, 1.5,
        1.6, 1.7, 1.8, 1.9, 2.0,
        2.1
}) (
    input en,
    input rst,
    output reg out
);
```

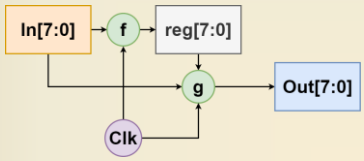


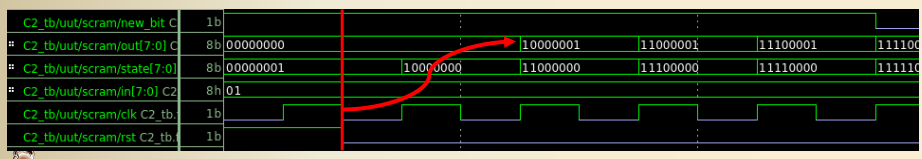
Counter.v

- WIDTH will be passed to determine the maximum of counter
- Out sets 1 when reaches its maximum value
- The counter increments on each clock cycle when in is 1 and out is 0
- The counter stops incrementing when it reaches its maximum value or is reset

Scrambler.v

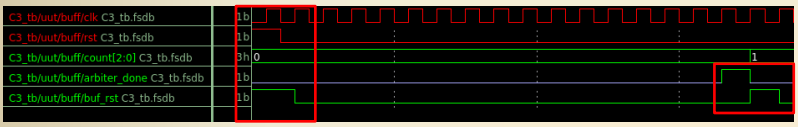
- Due to the presence of two clock-triggered functions, the output requires two clock cycles to be generated based on the corresponding input





Buffer.v

- The buffer reset is clock-triggered, so it will be delayed by one clock cycle compared to the system reset and arbiter done.



Outline

- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

Simulation Platform

- NC-Verilog
  - Cadence tool
  - Supports System-Verilog & Verilog file simulation
  - Generates .fsdb as waveform file
- CustomExplorer
  - Synopsys tool
  - Open .fsdb to visualize the waveform

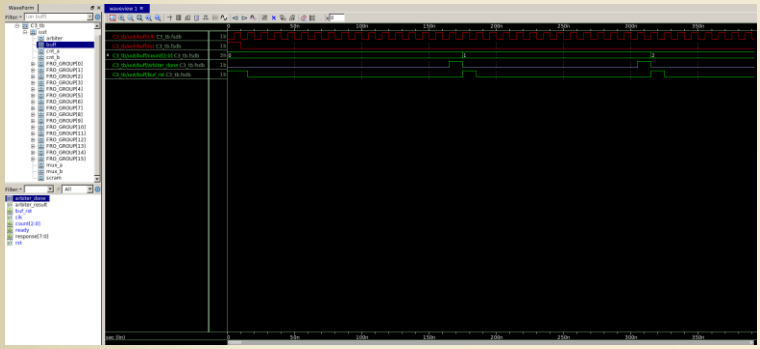
Makefile

- Simulation (make sim file=<testbench>)
  - It will use all .v files that do not contain the "tb" keyword in their filenames
  - You need to manually enter the full name of the testbench file
  - Your testbench file must include "tb" in its filename; otherwise, the Makefile will detect other testbench files
  - Your testbench must contain the following lines:
    - $fsdbDumpfile("<FileName>.fsdb");
    - $fsdbDumpvars(0, <ModuleName>);

    to generate the .fsdb file

Makefile

- View (make view file=<fsdb_file>)
  - It will visualize the .fsdb file generated by the simulation.



Outline

- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

Uniformity (UF)

- How evenly distributed the 1s and 0s are in the PUF response bits, 50% is the best
  - $Uniformity_i = \frac{1}{n}\sum_{j=1}^{n}(u_{i,j}) \times 100\%$
  - E.g., 10110101= $\frac{5}{8}$ = 62.5%
- You need to calculate your own simulated data with your testbenchs

Uniqueness (UQ)

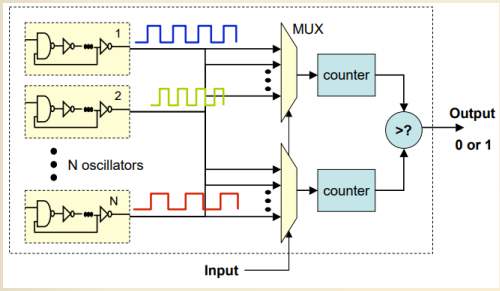
- The difference of a PUF's responses to the same challenge (C) when implemented on several PUF chips, 50% is the best
  - $Uniqueness = \frac{2}{k(k-1)}\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}\left(\frac{HD(S_i,S_j)}{n}\right) \times 100\%$
  - E.g., S1=10101100, S2=11001110, S3=11100100, HD(S1,S2) = 3, HD(S1,S3) = 2, HD(S2,S3) = 3…
  - UQ= $\frac{2}{3(3-1)}\frac{3+2+3}{8}$ = 33%
- You need to calculate your own simulated data with your testbenchs

Outline

- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

Bonus

- Implement a Ring Oscillator PUF (RO PUF) that differs from this architecture
- Other Scrambler



Outline

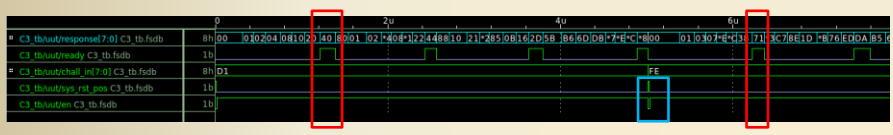
- Introduction & Background
- FRO PUF Architecture
- Verilog Implementation Details
- Testing & Simulation
- Evaluation & Analysis
- Bonus (Extra Challenges)
- Grading and Requirement

Grading

- Correctness(50%)
- The analysis of your PUF Design(30%)
- The report format and content(10%)
- Demo session(10%)
- Bonus(10%)

Correctness

- Only check the 1st ready response
- The same challenge produces the same response in the same PUF (Reliability)
- Test cases will be in a similar form as the public cases



The Analysis of PUF Design

- Explanation of Data Collection
- Presentation of Collected Data
- Interpretation of Results
- Justification of Observed Values

The Report

- Formatting & layout
- Clarity & organization
  - The use of visual elements (e.g., diagrams, table)
- Content quality

Demo

- Attendance(4%)
- Three Questions(2% * 3 = 6%)

Bonus

- Evaluating the advantages and disadvantages between two designs
- Conduct a complete analysis

Notice

- You have to put all .v files in a .zip file named StudID_PA1.zip (ex: 123456789_PA1.zip)
- A report named as StudID_Name_PA1.pdf
- Upload to ee-class

Wei-Hung, Lin

Student, Department of EE, NCU

Email: bb232399@g.ncu.edu.tw