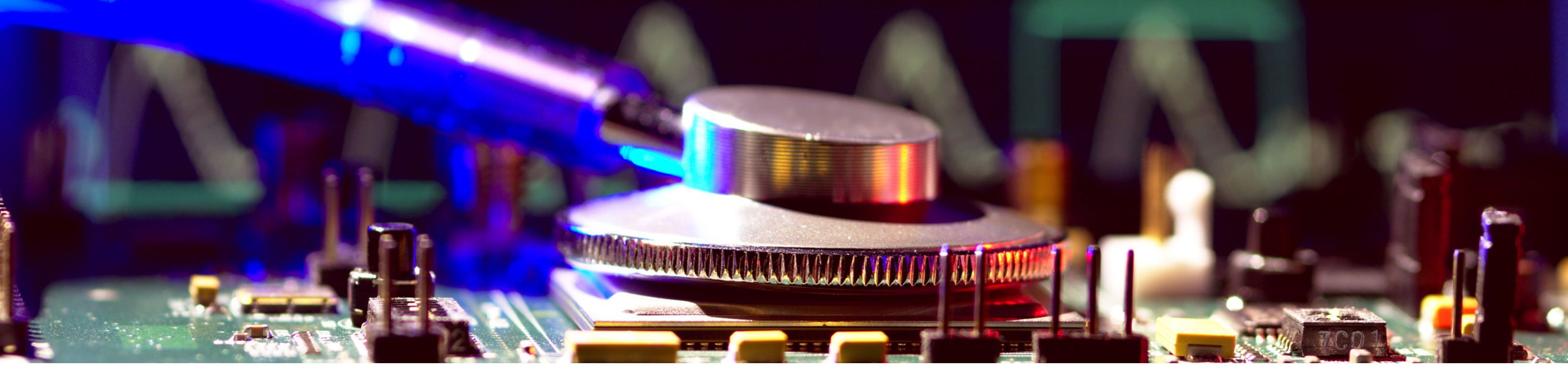


Flexible, Opensource workBench fOr Side-channel analysis (FOBOS)

FOBOS User Guide v2.0

Abubakr Abdulgadir **William Diehl** **Rajesh Velegalati**
Jens-Peter Kaps
[aabdulga, wdiehl, jkaps]@gmu.edu

March 26, 2018
George Mason University
Fairfax, Virginia

cryptography.gmu.edu

www.gmu.edu

Cryptographic Engineering Research Group
Department of Electrical and Computer Engineering George Mason University
3100 Engineering Building, 4400 University Drive, Fairfax, VA 22030-4444, USA
Voice: (703) 993-1561, Fax: (703) 993-1601

Contents

1 FOBOS Overview

1.1 Introduction

FOBOS is a side-channel analysis tool that is flexible, open-source and includes tools needed for power data acqusition and analysis. In this section we describe how to download and install FOBOS and describe the general procedure for using it to perform DPA attacks.

1.2 Requirements

1.2.1 Linux Requirements

The following items are requirements to running FOBOS on Linux machines:

1. Python need to be installed on the system. Installer can be downloaded from https://www.python.org.

2. Network configuration must allow the PC to be able to communicate to the Oscilloscope via IP network.

1.2.2 Windows Requirements

The following items are requirements to running FOBOS on Windows machines:

1. Python need to be installed on the system. Installer can be downloaded from https://www.python.org/.

2. Network configuration must allow the PC to be able to communicate to the Oscilloscope via IP network.

1.3 Installation

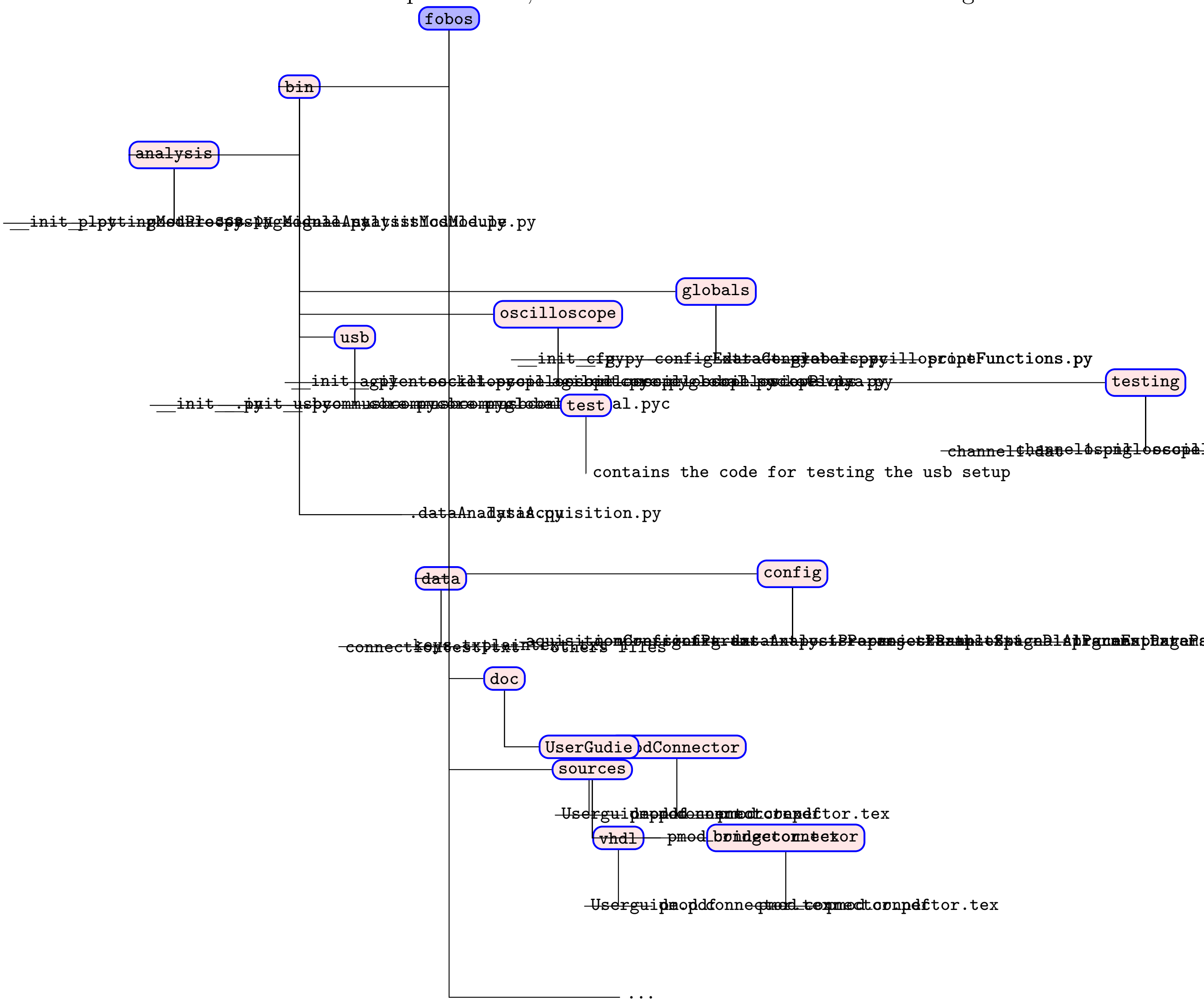
To install FOBOS software follow these steps:

1. Download FOBOS from https://cryptography.gmu.edu/fobos/.

2. Extract the file as follows:
   `tar xvzf fobos.tar.gz`

1.4 File Structure

After FOBOS is extracted from the compressed file, its file structure looks like the following:



1.5 Hardware Setup

FOBOS hardware consist of the following components:

1. FOBOS Control board.
2. FOBOS Victim board a.k.a DUT.
3. Control PC.
4. Power trace capturing device (e.g oscilloscope).

FOBOS control and victim boards are standard FPGA boards that need to be configured. FOBOS control board harware description is provided along with a victim wrapper. Victim cipher is user provided. FPGA boards need to be connected to other components. For details, please refer to Chapter 2.

1.6 Trace Acquisition

After FOBOS software and hardware are setup, encryptions can be run on the victim board and traces collected. This is refered to as Data Acquisition. To perform data acquisiton, two steps need to be done:

1. Edit the configuration file `$fobos/cofig/acquisitionConfig.txt`. Configuration parameters description provided in Chapter 3.

2. Run the dataAcquistion script as follows:
   ```
   cd $fobos
   python dataAcquisition.py
   ```

1.7 Data Analysis

Running DPA attack on the traces collected in the Data Aquisition phase is refered to as Data Analysis. Data Analysis uses two inputs; measured power traces and hypothetical power traces. To perform Data Analysis, Three steps need to be done:

1. Generate hypothetical power traces.

2. Edit configuration files at $fobos/config. Details on configuration parameters provided in Chapter 4.

3. Run the the dataAnalysis script as follows:
   ```
   cd $fobos
   python dataAnalysis.py
   ```

2 FOBOS Hardware Configuration

2.1 Oscilloscope Interface

Ocsilloscope is connected to the PC via IP netwok. IP configuration must be done on the oscilloscope for the PC to be able to collect traces. The oscillopscope used in tests is Agilent DSO6054A. To configure IP on this oscilloscope, please refer to vendor's documentation.

2.2 Crypto Algorithm Wrapper

The Victim Board runs the cipher that need to be attcked (user provided). A victim wrapper is included to facilitate communication between FOBOS Control Board and Victim Board. After the wrapper gets the data from the Control Board, it sends it to the victim cipher. After the victim cipher finishes, result is transfered back to the Control Board. Here we list signals used to interface between the victim wrapper and victim cipher:

1. clock : clock signal from the Control Board used to drive Victim Board.
2. data_to_crypto : A block of data to be processed by the victim cipher.
3. key_to_crypot : The key used by victim cipher.
4. data_out : result of victim cipher.
5. start : A handshake signal indcating the data_to_crypto and key_to_crypto are valid and instructs victim cipher to start processing.
6. done : A handshake signal generated by victim cipher to indcate that processing finished and data_out is valid result.

2.3 Control Board Programming

There are two generic parameters that need to be configured depending on the Control board used. The `Interface_Width` generic is the width of the bus used to communicate between the Control and Victim boards (in each direction). The `board` generic is used to indicate the type of control board used. These values are set in the `$fobos/sources/common/fobos-package.vhd` file. For valuse, refer to Table 2.1.

After the generics have been edited, the FPGA can now be programmed as follows:

1. Create a new project using Xilinx ISE. In the New Project wizard set the Project Settings per the Control Board used. Make sure to select the values for Family, Device, Package and Speed (See Fig 2.1 for an example).
2. From the Project menu, select Add Source... and add all files from `$fobos/sources/common`.

Table 2.1: Interface_Width Parameter

Control Board	Interface_Width	Board
Nexys2	16	1
Nexys3	4	2

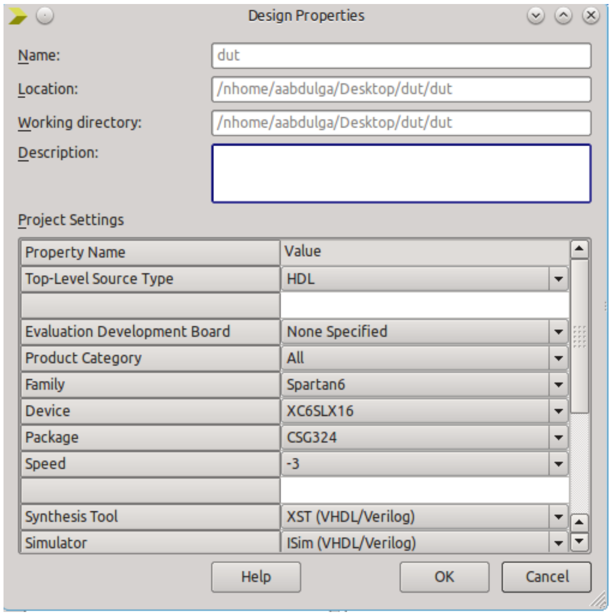


Figure 2.1: FOBOS Controller Desgin Properties

3. Repeat the previous process to add all vhdl files from `$fobos/sources/vhdl/control`. Also, add the appropriate UCF file depending on the Control Board used (See Fig 2.2) .

4. Set the `fobosControlTopLevel` module as the top-level module for this project (See Fig 2.3).

5. Generate the programming bit file for the control board by clicking ”Generate Programming File” in the Processes window.

6. Program the control board using Xilinx Impact. In the Processes window, click Configure Target Device (See Fig 2.4).

7. In the Impact window, click ”Boundary Scan” then form the File menu, click ”Initialize Chain” and assign the bit file to the FPGA. Now you may right-click the FPGA and click ”Program”.

2.4 DUT Board Programming

1. Create a new project using Xilinx ISE. In the New Project wizard set the Project Settings per the DUT Board used. Make sure to select the values for Family, Device, Package and Speed (See Fig 2.6 for an example).

2. From the Project menu select Add Source... and add all files from `$fobos/sources/common`.

3. Repeat the previous process to add all vhdl files from `$fobos/sources/vhdl/DUT` and make sure to add the appropriate UCF file depending on the DUT board used (See Fig 2.7).

4. Add the victim cipher vhdl files to the project (user provided).

5. Make sure to not use block RAMs in the implementation. .

   (a) Make sure to select the ”Implementation” view.

   (b) Right-click the Synthesize-XST process.

   (c) In the Preocess Properties window, select HDL Options and select ”Distributed” for the RAM Style property (See Fig 2.8).

   (d) Click OK.

6. Set the `dutTopLevel` as the top-level module in this project.

7. Generate the programming bit file for the DUT by clicking ”Generate Programming File” in the Processes window.

8. Program the DUT using Xilinx Impact. In the Processes window, click Configure Target Device (See Fig 2.10).

9. In the Impact window, click ”Boundary Scan” then form the File menu, click ”Initialize Chain” and assign the bit file to the FPGA. Now you may right-click the FPGA and click ”Program”.

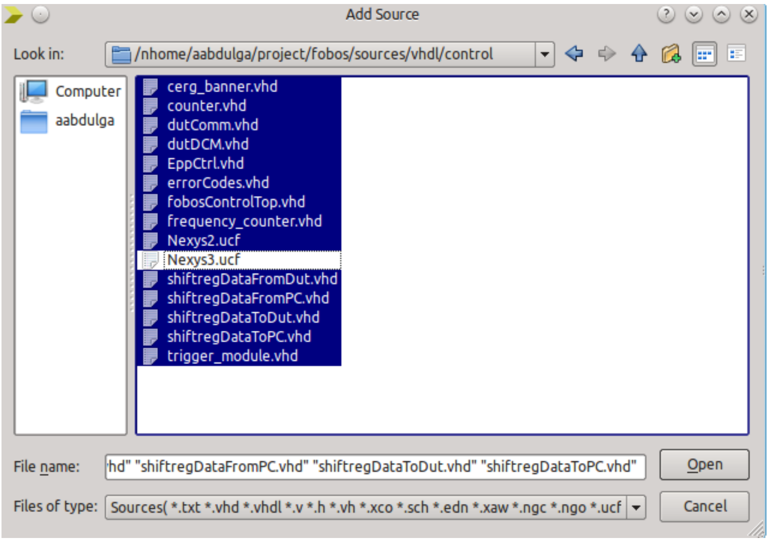


Figure 2.2: Adding Source Files to FOBOS Controller

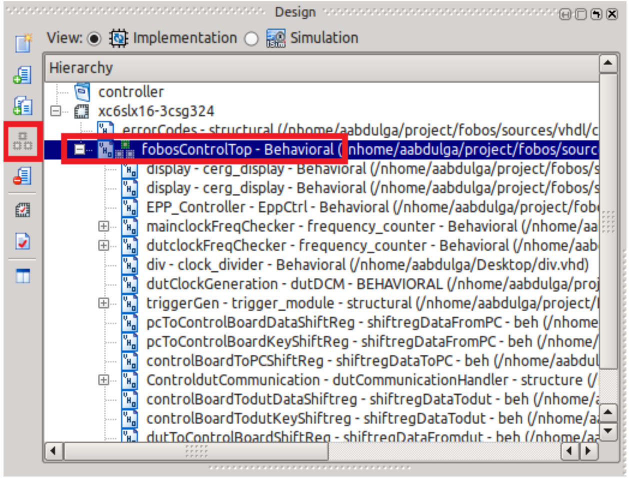


Figure 2.3: Setting Top-level Module

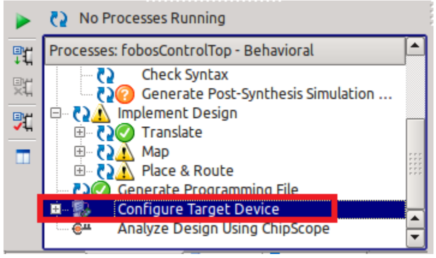


Figure 2.4:

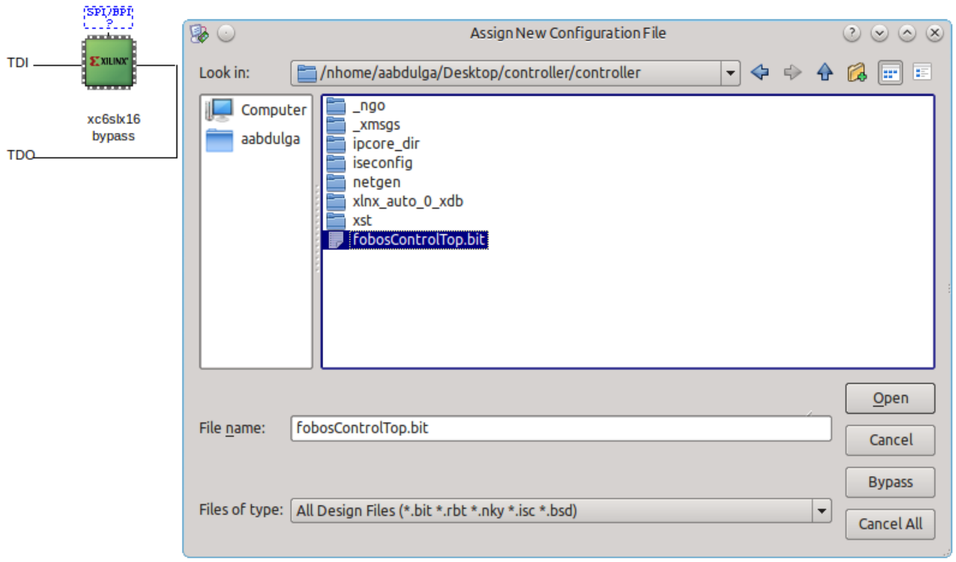


Figure 2.5: Progrmamming Control Board

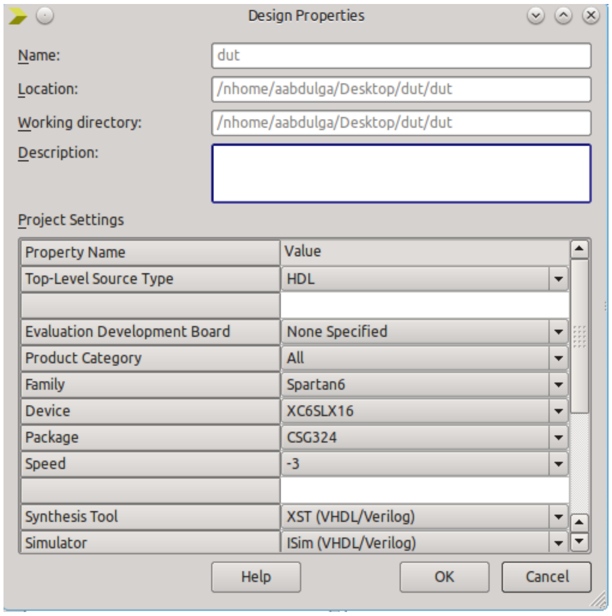


Figure 2.6: DUT Design Properties

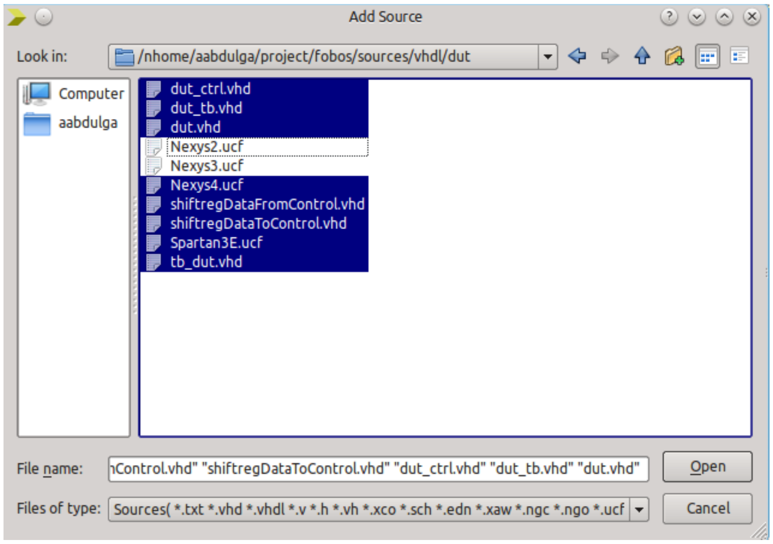


Figure 2.7: DUT Add Sources

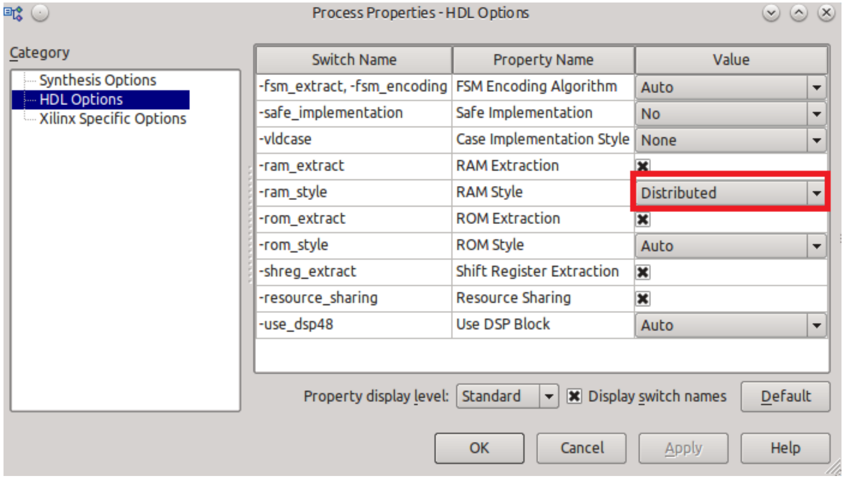


Figure 2.8: DUT RAM Style

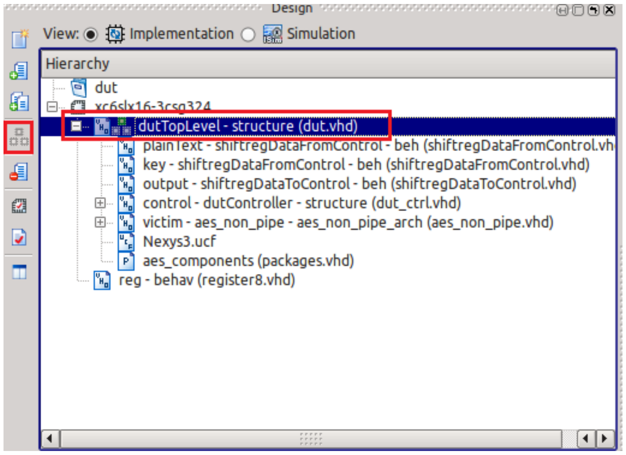


Figure 2.9: Set DUT Top-level

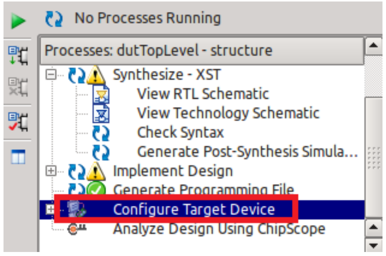


Figure 2.10:

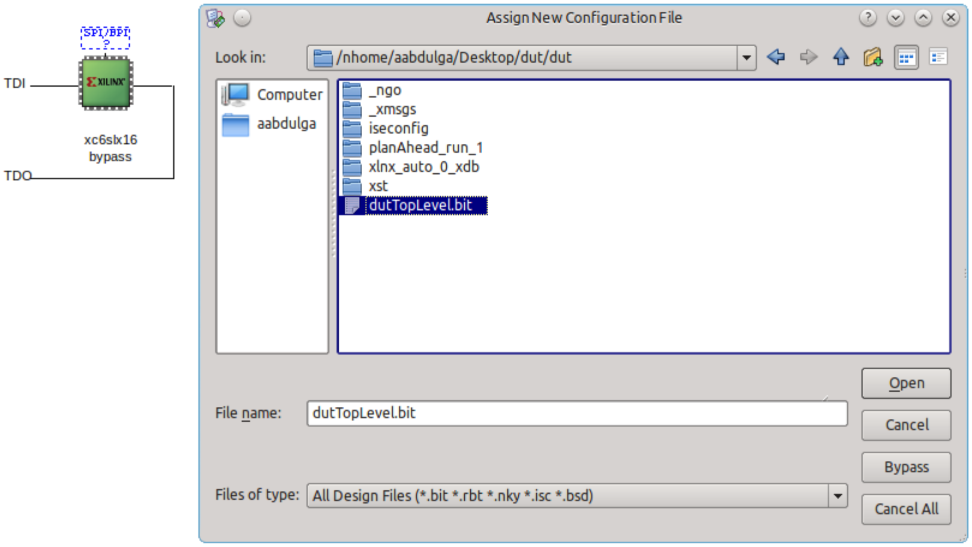


Figure 2.11: Progrmamming DUT

2.5 FOBOS Oscilloscope Configuration

Oscilloscope need to be conncted to the network and have an IP address and port number. To configure the oscilloscope, refer to vendors's documentation.

2.6 Connecting Hardware

To connect FOBOS hardware, follow these steps:

1. Connect control board to power and ground.
2. Connect control board tirgger output the oscilloscope channel.
3. Connect the control board to the DUT.
4. Connect the control board to the PC running the FOBOS software using USB.
5. Connect clock generator to control board. Set the clock generator to desired clock.
6. Connect the current probe to the oscilloscope.
7. Connect the current probe to the DUT's ground.
8. Connect DUT to power making sure that the current probe is measuring the current.
9. Connect DUT to power supply ground.

The following subsections ilustrate how to connect different FPGA boards as control and victim boards.

2.6.1 Nexys3 Control Board - Nexys4 DUT

Illustrated figure

2.6.2 Nexys3 Control Board - Nexys3 DUT

Illustrated figure

2.6.3 Nexys2 Control Board - Spartan3E DUT

Illustrated figure

3 FOBOS Data Acquisition

3.1 FOBOS Acquisition

Data Acquisition module is used to run encryptions on FOBOS hardware and collect traces. Plain text and key can be randomly generated or read from file. The output from this phase is a Numpy array file (.npy) that holds power traces.

3.2 Requirements

This module uses an osciloscpoe to measure and capture traces. Current version connects to oscilloscope using IP network. The acqusition module runs on a Windows or Linux PC and connected to the FOBOS control board via USB cable. FOBOS software and hardware need to be setup before Data Acqusition is run.

3.3 FOBOS Acquisition Configuration

Data Acquisition need to be configured. The configuration file is located at `$fobos/config/dataAcquisitionConfi` Table 3.1 lists all configuration parameter needed to configure Data Acquisition.

3.4 Running Data Acquisition

Data Acquisition can be run as follows:

```
cd $fobos
python dataAcquisition.py
```

Once this is done, the script will run encryptions on hardware, collect traces from the oscilloscope and save traces into a file. The script will create a new directory each time it runs. This directory is created in the project directory. Traces will be save in `$fobos/$workspace/$project/$attempt/Measurements/rawDataAligned.npy`. Where $attempt is a folder with a unique name created each time the script runs.

Table 3.1: acquisitionConfig.txt

Parameter	Possible Values	Description
Global Settings		
MEASUREMENT_FORMAT	dat	
LOGGING	INFO—DEBUG	Specifies log detail level. DEBUG is more verbose
CONTROL_BOARD	Nexys2—Nexys3	Control Board model
VICTIM_RESET	Integer (e.g 11)	Number of DUT clock cycles before resetting the DUT
TIME_OUT	Integer (e.g. 50000)	If DUT is not returning data after the number of clock cycles specified, a time out signal is sent to the PC
TRIGGER_WAIT_CYCLES	Integer (e.g 1)	Number of DUT clock cycles before sending trigger signal to oscilloscope
TRIGGER_LENGTH_CYCLES	Integer (e.g 1)	Number of DUT clock cycles the trigger signal remains high
PLAINTEXT_GENERATION	USER—RANDOM	Specifies if plain text is read from file or randomly generated
DATA_FILE	File name (e.g. plaintext.txt)	Specifies the plain text file name. File is saved in the sources directory
KEY_GENERATION	USER—RANDOM	Specifies if the key is read from file or randomly generated
KEY_FILE	File name (e.g keys.txt)	Specifies the key file name. File is saved in the sources directory
INPUT_FORMAT	hex	Specifies input data format. Currently, only hexadecimal is supported
OUTPUT_FORMAT	hex	Specifies output data format. Currently, only hexadecimal is supported
NUMBER_OF_ENCRYPTIONS_PER_TRACE	Integer (e.g 1)	Specifies the number of encryptions in each trace collected from the oscilloscope
BLOCK_SIZE	Integer (e.g 16)	Cipher block size in bytes
KEY_SIZE	Integer (e.g 16)	Cipher key size in bytes
DUMMY_RUN	YES—NO	
NUMBER_OF_TRACES	Integer (e.g. 2000)	Number of encryptions/traces to be done/collected
CAPTURE_MODE	MULTI—SINGLE	Specifies single encryption or multiple per trace
TRIGGER_THRESHOLD	Float (e.g 1.0)	Minimum trigger signal voltage to be considered a valid trigger
OSCILLOSCOPE	AGILENT—OPENADC	Oscilloscope model
OSCILLOSCOPE_IP	IP address (e.g 192.168.0.10)	Oscilloscope IP address
OSCILLOSCOPE_PORT	port number (e.g 5025)	Osciloscope port number

4 FOBOS Analysis

The Analysis module is used to perform DPA attack on power traces collected by the Acquisition module. In this section we provide description of the configuration and usage of this module. To perform analysis, to inputs are needed, the power traces and and hypothetical power model.

4.1 Power Model

Power model or hypothetical power data is user provided. FOBOS uses a text file for each key byte. This file includes a line for each key guess value (i.e. 0-255). Each line inculdes hypothetical power value for the specific key guess for all encryptions. Each value is an integer and separated from next value by a space. Fig 4.1 is an example for one key byte. The firs number is the estimated power when the byte equals zero for the frist encryption, first value in the second line is the estimated power when the key byte equals one for the first encryption and so on.

FOBOS expects to find these files at $fobos/data/. File names should be hypo-power-byte-BYTE_NUMBER.txt.

4.2 Analysis Configuration

Here we list all the configuration parameter used in the FOBOS Analysis and descriton of usage:

4.3 Running Data Analysis

Data Analysis can be run as follows:

```
cd $fobos
python dataAnalysis.py
```

Once this is done, the script reads the measured and hypothetical power data, runs DPA and produces several output files. The script prompts the user for the directory that contains the traces (the $attmpt dirctory) and then uses the power data in $attempt/Measurements as input. Once user specifies the directory it is save in $fobos/config/projectPath.txt and used in subsequent runs without prompting the user. To make FOBOS ask for the $attempt directory, remove this file. The script will create a new directory each time it runs. This directory is created in the project directory. Output files will be save in `$fobos/$workspace/$project/$attempt/analysis/$analysis-attempt/`. Where $analysis-attempt is a folder with a unique name created each time the script runs.

[frame=single] 4 6 3 4 5 1 5 4 4 2 4 3 5 6 3 4 4 3 4 2 . . . . 3 2 7 6 5 6 5 5 7 3 5 4 4 2 6 5 2 3 2 5 . . . . 5 4 3 6 4 5 3 4 3 3 6 5 3 0 2 5 6 5 6 5 . . . .

Figure 4.1: Hypothetical Power Model

Table 4.1: signalAlignmentParams.txt

Parameter	Possible Values	Description
CAPTURE_MODE	MUL—SINGLE	Use SINGLE when each trace captured by the oscilloscpoe contains one trace and MULTI when the traces contains multiple encryptions
TRIGGER_THRESHOLD	Float (e.g. 1.0)	Floating point number represnting minimum voltage to indicate a valid trigger.

Table 4.2: samplesSpacesDisp.txt

Parameter	Possible Values	Description
SAMPLE_WINDOW_SIZE	Integer (e.g 2000)	Number of samples per trace to be used in analysis.
SAMPLE_WINDOW_START	Integer (e.g 100)	The number of the first sample in the window.

Table 4.3: compressionParams.txt

Parameter	Possible Values	Description
COMPRESSION_LENGTH	Integer (e.g. 10)	Number of samples to be compressed into one samples.
COMPRESSION_TYPE	MAX—MIN—MEAN	The operation to be performed to generated the compressed sample.

Table 4.4: traceExpunge.txt

Parameter	Possible Values	Description
TRACE_EXPUNGE_PARAMS	STD—VAR:BELOW:ABOVE—NO (e.g. VAR:0.0004:0.00045)	Specifies the operation to be done on traces and the lower/upper bounds of traces not to discard.

Table 4.5: postProcessesParams.txt

Parameter	Possible Values	Description
SAMPLE_SPACE_DISPOSITION		
COMPRESS_DATA	1-3—NO	
TRACE_EXPUNGE	1-3—NO	

Table 4.6: projectPath.txt

Parameter	Possible Values	Description
N/A	A director path (e.g. /fobos/workspace/tests/1-test)	This file contains the path to the directory that contains data to be analysed. If this file is missing, FOBOS will prompt user to select the directory and save user select to this file.

5 Function Descriptions

5.1 FOBOS - Analysis Module

FOBOS's analysis module uses a set of python scripts to post process the raw measurement data obtained from the oscilloscope and perform analysis on the obtained data.Various functions implemented in the Analysis module is described below:

5.2 FOBOS - Capture Module

The Capture module is used to run encryptions on hardware and capture traces from the ocsilloscope.

5.3 FOBOS - Other Functions

Here we list helper functions that are used by the Analysis and Capture modules.

Table 5.1: getAlignedMeasuredPowerData Function

teal**signalAnalysisModule.getAlignedMeasuredPowerData**	
Usage	`signalAnalysisModule.getAlignedMeasuredPowerData()`
Inputs	None
Outputs	M x N Numpy matrix that holds aligned traces. Where M is the number of traces and N is the number of samples per trace.
Description	Reads aligned traces from the $Workspace/$projectName/$attempt/Measurements directory and loads it to an M x N Numpy matrix. This function calls signalAnalysisModule.readAlignedDataFromFile().

Table 5.2: readAlignedDataFromFile Function

teal**signalAnalysisModule.readAlignedDataFromFile**	
Usage	`signalAnalysisModule.readAlignedDataFromFile()`
Inputs	None
Outputs	M x N Numpy matrix that holds aligned traces. Where M is the number of traces and N is number of samples per trace.
Description	Reads aligned data from file and returns M x N Numpy matrix.

Table 5.3: detectSampleSize Function

teal**signalAnalysisModule.detectSampleSize**	
Usage	`signalAnalysisModule.detectSampleSize(fileName)`
Inputs	Aligned traces file name.
Outputs	Number of samples in a trace.
Description	Reads the first 10 traces and returns the number of samples in the largest trace. If the number of traces is less than 10 all traces are read. This is done to be able to adjust all traces to the same number of samples if some do not have the same size due to acquisition timing.

Table 5.4: adjustSampleSize Function

tealsignalAnalysisModule.adjustSampleSize	
Usage	`signalAnalysisModule.adjustSampleSize(sampleLength, dataArray)`
Inputs	• sample length to adjust to. • N x 1 nympy array that represents one trace where N is the number of samples in the trace.
Outputs	SampleLenght x 1 Numpy array that represents the adjusted trace.
Description	Used to modify the number of samples in a trace. If the number of samples is less than SampleLength, the array is padded with zeros. If the number of samples is more than SampleLength, the array is truncated. The function does nothing if the number of samples is equal to SampleLength.

Table 5.5: acquirePowerModel Function

tealsignalAnalysisModule.acquirePowerModel	
Usage	`signalAnalysisModule.acquirePowerModel (HypotheticalDataFileName,globals.ADAPTIVE_CPA)`
Inputs	• Power model file name • Correlation type
Outputs	M x N Numpy array that holds the hypothetical power traces. Where N is the number of encryptions/decryptions and M is the number of key guesses.
Description	Reads the hypothetical power data from file in $fobos/data. Returns M x N Numpy array that holds the hypothetical power traces. Where N is the number of encryptions/decryptions and M is the number of key guesses.

Table 5.6: updatePowerModelUsingBCPA Function ***********revise

teal**signalAnalysisModule.updatePowerModelUsingBCPA**	
Usage	`signalAnalysisModule.updatePowerModelUsingBCPA(array1, array2, rowA)`
Inputs	• Array1 • Array2 • rowA
Outputs	
Description	Called from signalAnalysisModule.acquirePowerModel().

Table 5.7: computeAlignedData Function

teal**signalAnalysisModule.computeAlignedData**	
Usage	`signalAnalysisModule.computeAlignedData(powerData, triggerData)`
Inputs	• N x 1 Numpy array that holds measured power data. Where N is the number of samples. • N x 1 Numpy array that holds trigger power data. Where N is the number of samples.
Outputs	Aligned power trace (K x 1 Numpy array where K is the number of samples).
Description	Uses powerData and triggerData to generate the aligned trace. The function looks for the rising edge of the trigger signal to determine the start of the trace.

Table 5.8: correlation_pearson Function

tealsca.correlation_pearson	
Usage	`sca.correlation_pearson(measuredPowerData, hypotheticalPowerData)`
Inputs	• M x N Numpy matrix for power traces. Where M is the number of encryptions/decryptions, N the number of samples per trace. • L x M array the represents the hypothetical power values. Where M is the number of encryptions/decryptions and L is the number of key guesses (i.e for byte guess, L=256).
Outputs	N x L Numpy correlation matrix where N is the number of samples per trace.
Description	Calculates Pearson correlation between the hypothetical data and measured data. Returns an N x L correlation matrix. When we guess byte values L = 256.

Table 5.9: findMinimumGuessingEntropy Function*********revise

tealsca.findMinimumGuessingEntropy	
Usage	`signalAnalysisModule.computeAlignedData(measuredPowerData, triggerData)`
Inputs	• measured power data. • hypotheticalPowerData.
Outputs	
Description	

Table 5.10: calculate_autocorrelation Function ***** revise

tealsca.calculate_autocorrelation	
Usage	`sca.calculate_autocorrelation(alignedData)`
Inputs	M x N Numpy matrix that holds aligned power data. Where M is the number of traces and N is the number of samples per trace.
Outputs	
Description	

Table 5.11: plotTrace Function

teal**plottingModule.plotTrace**	
Usage	`plottingModule.plotTrace(dataToPlot, traceNos, plotType)`
Inputs	• M x N Numpy matrix that holds traces to plot. Where M is the number of traces and N is the number of samples. • Trace number • Plot type per trace. The traces to be plotted.
Outputs	None
Description	Plots the traces represented by the Numpy matrix. The x-axis represents time and the y-axis represents voltage.

Table 5.12: plotHist Function

teal**plottingModule.plotHist**	
Usage	`plottingModule.plotHist(corrMatrix, corrType)`
Inputs	• M x N Correlation Numpy matrix. • Correlation type.
Outputs	None
Description	Plots a histogram. The x-axis represents the key guess and the y-axis represenst the number of occurrences.

Table 5.13: plotCorr Function

teal**plottingModule.plotCorr**	
Usage	`plottingModule.plotCorr(corrMatrix, corrType)`
Inputs	• M x N Numpy correlation matrix • Correlation type
Outputs	None
Description	Plots correlation data.

Table 5.14: sampleSpaceDisp Function

tealpostProcessingModule.sampleSpaceDisp	
Usage	`postProcessingModule.sampleSpaceDisp(alignedData)`
Inputs	M x N Numpy matrix that holds aligned data. Where M is the number of traces and N is the number of samples per trace.
Outputs	M x Window_Size Numpy array that holds the aligned data after removing samples
Description	Removes samples before WINDOW_START and after WINDOW_START + WINDOW_SIZE - 1 from each trace.

Table 5.15: compressData Function

tealcompressData	
Usage	`compressData(measuredPowerData)`
Inputs	M x N Numpy array that represents traces. Where M is the number of traces and N is the number of samples per trace.
Outputs	M x K Numpy array that represents compressed traces. Where M is the number of traces and $K = \frac{N}{COMPRESSION\_LENGTH}$.
Description	Summarizes COMPRESSION_LENGTH samples into one sample. The summarization type depends on the COMPRESSION_TYPE configuration parameter which can be MEAN, MIN or Max.

Table 5.16: compress Function

tealpostProcessingModule.compress	
Usage	`postProcessingModule.compress (a, compressionLenght, compressionType)`
Inputs	• M x N Numpy array that represents traces. Where M is the number of traces and N is the number of samples per trace. • Number of samples to compress into one sample. • Compression type.
Outputs	M x K Numpy array that represents compressed traces. Where M is the number of traces and $K = \frac{N}{COMPRESSION\_LENGTH}$.
Description	This function is called by postProcessingModule.compressData() to do the compression.

Table 5.17: traceExpunge Function

tealpostProcessingModule.traceExpunge	
Usage	`postProcessingModule.traceExpunge(measuredPowerData)`
Inputs	M x N Numpy array that represents traces. Where M is the number of traces and N is the number of samples per trace.
Outputs	L x N Numpy array that represents traces after removing the traces that do not fall in acceptable range. Where L is the number of traces and N is the number of samples per trace.
Description	Removes traces that do not fall in acceptable range.

Table 5.18: openOscilloscopeConnection Function

tealOscilloscope_core.openOscilloscopeConnection	
Usage	`Oscilloscope_core.openOscilloscopeConnection()`
Inputs	None
Outputs	None
Description	Connects to oscilloscope. It opens a socket using the IP address OSCILLOSCOPE_IP and port number OSCILLOSCPOE_PORT. Also gets the oscilloscope identifier.

Table 5.19: setOscilloscopeConfigAttributes Function

tealOscilloscope_core.setOscilloscopeConfigAttributes	
Usage	`Oscilloscope_core.setOscilloscopeConfigAttributes()`
Inputs	None
Outputs	None
Description	Configures the oscilloscope by sending commands (in text format) to the oscilloscope.

Table 5.20: initializeOscilloscopeDataStorage Function

tealOscilloscope_core.initializeOscilloscopeDataStorage	
Usage	`Oscilloscope_core.initializeOscilloscopeDataStorage()`
Inputs	None
Outputs	None
Description	Creates empty Numpy arrays for each enabled oscilloscope channel.

Table 5.21: armOscilloscope Function

tealOscilloscope_core armOscilloscope	
Usage	`Oscilloscope_core.armOscilloscope()`
Inputs	None
Outputs	None
Description	Instructs the oscilloscope to digitize channels specified in FOBOS configuration.

Table 5.22: populateOscilloscopeDataStorageAndAlign Function

teal**Oscilloscope_core.populateOscilloscopeDataStorageAndAlign**	
Usage	`Oscilloscope_core.populateOscilloscopeDataStorageAndAlign(traceCount)`
Inputs	The number of current trace.
Outputs	None
Description	Reads power data trace from oscilloscope and trigger signal trace. It then aligns the trace to the trigger signal and saves the aligned trace to file.

Table 5.23: closeOscilloscopeConnection Function

teal**Oscilloscope_core.closeOscilloscopeConnection**	
Usage	`Oscilloscope_core.closeOscilloscopeConnection()`
Inputs	None
Outputs	None
Description	Closes socket that connects to oscilloscope.

Table 5.24: openControlBoardConnection Function

teal**Oscilloscope_core.openControlBoardConnection**	
Usage	`usbcomm_core.openControlBoardConnection()`
Inputs	None
Outputs	None
Description	Initializes connection to control board, resets control board and reads control board and victim clocks.

Table 5.25: initializeControlBoardConnection Function

teal**usbcomm_core.initializeControlBoardConnection**	
Usage	`usbcomm_core.initializeControlBoardConnection()`
Inputs	None
Outputs	None
Description	Initializes the USB connection to the board. Called from OpenControlBoardConnection().

Table 5.26: sendTriggerParamsToControlBoard Function

teal**usbcomm_core.sendTriggerParamsToControlBoard**	
Usage	`usbcomm_core.sendTriggerParamsToControlBoard()`
Inputs	None
Outputs	None
Description	Sends the trigger parameters to the control boards. Parameters are: TRIGGER_WAIT_CYCLES and TRIGGER_LENGTH_CYCLES.

Table 5.27: runEncrytionOnControlBoard Function

tealusbcomm_core.runEncrytionOnControlBoard	
Usage	`usbcomm_core.runEncrytionOnControlBoard(traceCount)`
Inputs	The number of block used in encryption.
Outputs	None
Description	Sends a block of data to control board to do encryption. The key is sent before sending the frist block.

Table 5.28: sendKeyToControlBoard Function

tealusbcomm_core.sendKeyToControlBoard()	
Usage	`usbcomm_core.sendKeyToControlBoard()`
Inputs	None
Outputs	None
Description	Sends the key to control board. This function is called from usbcomm_core.runEncrytionOnControlBoard().

Table 5.29: sendBlockOfDataToControlBoard Function

tealusbcomm_core.sendBlockOfDataToControlBoard	
Usage	`usbcomm_core.usbcomm_core.sendBlockOfDataToControlBoard (traceCount)`
Inputs	The number of block used in encryption
Outputs	None
Description	Sends a block of data to the control board. This function is called from usbcomm_core.runEncrytionOnControlBoard().

Table 5.30: saveControlBoardOutputDataStorage Function

tealusbcomm_core_core.saveControlBoardOutputDataStorage	
Usage	`Oscilloscope_core.saveControlBoardOutputDataStorage()`
Inputs	None
Outputs	None
Description	Saves output from control board (cipher text) to file. File is stored in $Workspace/$projectName/$attempt/output/

Table 5.31: getKeyForAnalysis Function

tealdataGenerator.getKeyForAnalysis	
Usage	`dataGenerator.getKeyForAnalysis()`
Inputs	None
Outputs	Key formated as a list of hexadecimal bytes.
Description	Reads the key from file in $Workspace/$project/$attempt/output/ directory.

Table 5.32: getPlainText Function

tealdataGenerator.getPlainText	
Usage	`dataGenerator.getPlainText()`
Inputs	None
Outputs	A list of blocks that represents plain text.
Description	Generates random plain text or reads from file depending on configuration. Plain text file is located in $fobos/$sources/ directory.

Table 5.33: generateRandomKey Function

tealdataGenerator.generateRandomKey	
Usage	`dataGenerator.generateRandomKey()`
Inputs	None
Outputs	A list of key bytes. Each byte is represented as a hexadecimal string.
Description	Generates a random key in hexadecimal format. Key size is read form the KEY_SIZE configuration parameter.

Table 5.34: convertToHex Function**********revise

tealdataGenerator.convertToHex	
Usage	`dataGenerator.convertToHex(hexString)`
Inputs	Hexadecimal string
Outputs	
Description	

Table 5.35: configureWorkspace Function

tealconfigExtract.configureWorkspace()	
Usage	`configExtract.configureWorkspace()`
Inputs	None
Outputs	None
Description	Creates the project directory in the workspace and creates directories to store measured power data, cipher text and plain text etc. It also copies some configuration files and other files into the project directory.

Table 5.36: extractConfigAttributes Function

tealconfigExtract.extractConfigAttributes()	
Usage	`configExtract.extractConfigAttributes()`
Inputs	None
Outputs	None
Description	Reads the main configuration file to get configuration attributes. It also reads the acquisition configuration file and extracts configuration attributes.

Table 5.37: updatePowerAndTriggerFileNames Function

tealconfigExtract.updatePowerAndTriggerFileNames	
Usage	`configExtract.updatePowerAndTriggerFileNames()`
Inputs	None
Outputs	None
Description	Checks for the existance of measured data files and trigger data file and sets variables to the file names.

Table 5.38: configureAnalysisWorkspace Function

tealconfigExtract.configureAnalysisWorkspace	
Usage	`configExtract.configureAnalysisWorkspace()`
Inputs	None
Outputs	None
Description	Configures the analysis workspace directory by creating directories to store analysis results and copies configuration files.

Table 5.39: extractAnalysisConfigAttributes Function

tealconfigExtract.extractAnalysisConfigAttributes	
Usage	`configExtract.extractAnalysisConfigAttributes(fileName)`
Inputs	Configuration file name.
Outputs	None
Description	Reads the file provided and gets configuration attributes. Also, copies the file to the projects local configuration directory for future reference.

Table 5.40: goToSleep Function

tealsupport.goToSleep	
Usage	`support.goToSleep(value)`
Inputs	Time to sleep in seconds.
Outputs	None
Description	Sleep for number of seconds.

Table 5.41: exitProgram Function

tealsupport.exitProgram	
Usage	`support.exitProgram()`
Inputs	None
Outputs	None
Description	Self-explanatory

Table 5.42: wait Function

teal**support.wait**	
Usage	`support.wait()`
Inputs	None
Outputs	None
Description	Waits for the user to press Enter to continue program execution.

Table 5.43: clear_screen Function

teal**support.clear_screen**	
Usage	`support.clear_screen()`
Inputs	None
Outputs	None
Description	Self-explanatory.

Table 5.44: convertToByteArray Function

teal**support.convertToByteArray**	
Usage	`support.convertToByteArray(hexString)`
Inputs	Hexadecimal string
Outputs	A byte array
Description	Converts a hexadecimal string to a byte array.

Table 5.45: arrayToString Function

teal**support.arrayToString(array)**	
Usage	`support.arrayToString(array)`
Inputs	Array to convert
Outputs	A string that consist of array elements
Description	Self-explanatory.

Table 5.46: readFile Function

teal**support.readFile**	
Usage	`support.readFile(fileName)`
Inputs	File name
Outputs	A string that holds file content.
Description	Self-explanatory.

Table 5.47: removeFile Function

teal**Support.removeFile**	
Usage	`Support.removeFile(fileName)`
Inputs	File name
Outputs	None
Description	Self-explanatory.

Table 5.48: removeComments Function

teal**Support.removeComments(datalist)**	
Usage	`Support.removeComments(datalist)`
Inputs	Data list
Outputs	???
Description	Removes comments (anything after a '#' sign) from a list of strings.

A Maybe Something